Spatial R course - 3

Tobias Andermann

5/25/2020

Accessing biodiveristy data through web services (GBIF)

In this tutorial we will go through some different functions related to downloading biodiveristy data, mostly from GBIF. The point of this tutorial is to provide you the tools to properly download such data in publication quality (DOI-assigned and thus tracable) and to provide you with some handy plotting functions to display the downloaded data.

Dependencies:

```
library(sp)
library(rgbif)
library(ggplot2)
# don't worry if you can't install mapr and/or RColorBrewer, these packages are o
nly necessary for a little extra at the end of the tutorial, but not really essen
tial for this tutorial
library(mapr)
library(RColorBrewer)
```

1. Define taxonomy

Taxonomy can be a tricky topic. Several different names exist for many taxa, variations being caused by misspellings, different synonyms and regional differences in common names for species. If you want to extract all records for a certain taxon, you first needs to define a coherent taxonomy and in the worst case you need to sync all available datasets to this chosen taxonomy. This can be a rabbit-hole that can make the collection of large datasets from public databases very time consuming. Luckily GBIF is working with one consistent taxonomy, which most records are assigned to. In this tutorial we will work with this GBIF backbone taxonomy (NUB). Check out the description of the GBIF taxonomy under this link (https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c#description) to understand how this taxonomy is derived and how to cite it.

The following identifier points to the NUB taxonomy, which will make more sense in a little bit.

```
nub <- 'd7dddbf4-2cf0-4f39-9b2a-bb099caae36c'
```

2. Pick a species/genus/family

Pick your own species, genus, or family of interest, for which you want to extract occurrence data from GBIF. If you are feeling sufficiently familiar with R, go ahead and extract occurrence data for multiple taxa, which at the end will yield the most interesting results when plotting the data. In this tutorial we will use an example species but it's strongly encouraged for you to go through the exercise with your own picked taxon (doens't have to be a species, can be a smaller or larger taxonomic entity).

```
taxon_name <- "Turdus merula"
```

There is a useful search function in rgbif called `name_suggest()`. This will return any matches with your provided taxon name and return the name as well as the taxonomic rank of the match:

```
library(rgbif)
name_suggest(q=taxon_name)
```

```
## # A tibble: 13 x 3
##        key canonicalName              rank
##      <int> <chr>                      <chr>
##  1 2490719 Turdus merula              SPECIES
##  2 6094911 Turdus merula sowerbyi     SUBSPECIES
##  3 9173280 Turdus merula nigropileus  SUBSPECIES
##  4 6094902 Turdus merula mandarinus   SUBSPECIES
##  5 6094954 Turdus merula cabrerae     SUBSPECIES
##  6 8917151 Turdus merula mallorcae    SUBSPECIES
##  7 9095250 Turdus merula buddae       SUBSPECIES
##  8 6094935 Turdus merula syriacus     SUBSPECIES
##  9 6094947 Turdus merula mauritanicus SUBSPECIES
## 10 5846244 Turdus merula intermedius  SUBSPECIES
## 11 6094940 Turdus merula aterrimus    SUBSPECIES
## 12 6171845 Turdus merula merula       SUBSPECIES
## 13 6094960 Turdus merula azorensis    SUBSPECIES
```

3. Check taxonomic information

Now we use the `name_lookup()` function of the rgbif package to check if our picked taxon exists in the chosen GBIF backbone taxonomy and what information is stored with it. In order for the function to find our taxon in the taxonomy, we need to provide the rank that the taxon name represents (is it a subspecies, species, genus, or family name?). We can extract the correct rank classification from the results of the `name_suggest()` function as shown above. In this example we're working with a taxon name that is on the species level. Accepted ranks are: CLASS, CULTIVAR, CULTIVAR_GROUP, DOMAIN, FAMILY, FORM, GENUS, INFORMAL, INFRAGENERIC_NAME, INFRAORDER, INFRASPECIFIC_NAME, INFRASUBSPECIFIC_NAME, KINGDOM, ORDER, PHYLUM, SECTION, SERIES, SPECIES, STRAIN, SUBCLASS, SUBFAMILY, SUBFORM, SUBGENUS, SUBKINGDOM, SUBORDER, SUBPHYLUM, SUBSECTION, SUBSERIES, SUBSPECIES, SUBTRIBE, SUBVARIETY, SUPERCLASS, SUPERFAMILY, SUPERORDER, SUPERPHYLUM, SUPRAGENERIC_NAME, TRIBE, UNRANKED, VARIETY If you are uncertain about how to parse your taxon name into this function, check the helpfunction by executing `?name_lookup()` in R. Note that in the command below we are using the `datasetKey=nub` settings, which is the GBIF standard taxonomy we defined earlier.

```
rank = 'species'
taxon_taxonomy_data = name_lookup(query=taxon_name, rank=rank, datasetKey=nub, li
mit=1)
taxon_taxonomy_data
```

```
## $meta
## # A tibble: 1 x 4
##   offset limit endOfRecords count
##    <int> <int> <lgl>        <int>
## 1      0     1 FALSE           15
##
## $data
## # A tibble: 1 x 34
##      key scientificName datasetKey constituentKey parentKey parent kingdom
##    <int> <chr>          <chr>      <chr>              <int> <chr>  <chr>
## 1 2.49e6 Turdus merula… d7dddbf4-… 7ddf754f-d193…   2490714 Turdus Animal…
## # … with 27 more variables: phylum <chr>, order <chr>, family <chr>,
## #   genus <chr>, species <chr>, kingdomKey <int>, phylumKey <int>,
## #   classKey <int>, orderKey <int>, familyKey <int>, genusKey <int>,
## #   speciesKey <int>, canonicalName <chr>, authorship <chr>,
## #   publishedIn <chr>, nameType <chr>, taxonomicStatus <chr>, rank <chr>,
## #   origin <chr>, numDescendants <int>, numOccurrences <int>,
## #   extinct <lgl>, habitats <chr>, nomenclaturalStatus <lgl>,
## #   threatStatuses <chr>, synonym <lgl>, class <chr>
##
## $facets
## NULL
##
## $hierarchies
## $hierarchies$`2490719`
##   rankkey          name
## 1       1      Animalia
## 2      44      Chordata
## 3     212          Aves
## 4     729 Passeriformes
## 5    5290      Turdidae
## 6 2490714        Turdus
##
##
## $names
## $names$`2490719`
##                            vernacularName language
## 1                                   Amsel      deu
## 2                              blackbird      eng
## 3                                  merel      nld
## 4                             merle noir      fra
## 5                              blackbird      eng
## 6                     Eurasian Blackbird      eng
## 7                             Merle noir      fra
## 8                       Common Blackbird      eng
## 9                     Eurasian Blackbird      eng
## 10 Eurasian Blackbird / Common Blackbird      eng
## 11                      Common Blackbird      eng
## 12                    Eurasian Blackbird      eng
## 13                            Merle noir      fra
## 14                      common blackbird      eng
## 15                                  Amsel      deu
## 16                      common blackbird      eng
## 17                             karatavuk      tur
```

```
## 18                          merel      nld
## 19                     merle noir      fra
## 20                        mulleja      sqi
## 21                        mullija      sqi
## 22                       mullizeza     sqi
## 23                       mwyalchen     cym
## 24                        mëllënja     sqi
## 25                   qofka e murrme     sqi
## 26                           qukla      sqi
## 27                      svarttrost      nor
## 28                  Κοινός Κότσυφας     ell
## 29                             Кос      bul
## 30                           Amsel
## 31                Eurasian blackbird
## 32                       blackbird
## 33                           Amsel      deu
## 34                       Blackbird      eng
## 35                     Mirlo Común      spa
## 36                     Mustarastas      fin
## 37                      Merle noir      fra
## 38                     Fekete rigó      hun
## 39                           Merlo      ita
## 40                           Merel      nld
## 41                             Kos      pol
## 42                     Melro-preto      por
## 43                        Koltrast      swe
## 44                         Solsort      dan
## 45                           Amsel      deu
## 46                  Common Blackbird      eng
## 47                       Kvørkveggja      fao
## 48                       Mustarastas      fin
## 49                           Solsort      dan
## 50                       Svarttrost      nob
## 51                     Svartßröstur      isl
## 52                          koltrast      swe
## 53                           Merel      nld
## 54                    Zwarte lijster      nld
## 55                  Common Blackbird      eng
## 56                Eurasian Blackbird      eng
## 57                       merle noir      fra
## 58                           Amsel      deu
## 59                  Common Blackbird      eng
## 60                Eurasian Blackbird      eng
## 61                       merle noir      fra
## 62                           Amsel      deu
## 63                  Common Blackbird      eng
## 64                           Merel      nld
## 65                      Merle noir      fra
## 66                           Merlo      ita
## 67                     Mirlo Común      spa
## 68                         Solsort      dan
## 69                       Svarttrost      nor
## 70                     drozd čierny      slk
## 71                      fekete rigó      hun
## 72                  juodasis strazdas      lit
```

```
## 73                              koltrast      swe
## 74                                   kos      slv
## 75                       kos (zwyczajny)      pol
## 76                             kos černý      ces
## 77                   melnais mežastrazds      lav
## 78                                 melro      por
## 79                                 merla      cat
## 80                            mustarastas      fin
## 81                             musträstas      est
## 82                          Чёрный дрозд      rus
## 83                                   乌鸫      zho
## 84                                   烏鶫      zho
```

You can see there is a lot of useful data stored in this taxonomy. First we can extract the **numerical taxon id** which we will use in following steps to extract occurrence records for this taxon. The advatange of using a numerical id is that the taxon is unmistakenly defined and will not anymore be subject to misspellings and different synonyms from here on.

```
taxon_id = taxon_taxonomy_data$data$key
taxon_id
```

```
## [1] 2490719
```

The `name_lookup()` function also provides us the taxon ids of the encompassing taxa higher up in the taxonomic hierarchy. In this example it tells us that the blackbird belongs to the genus Turdus in the family Turdidae etc.

```
taxon_taxonomy_data$hierarchies
```

```
## $`2490719`
##   rankkey          name
## 1       1      Animalia
## 2      44      Chordata
## 3     212          Aves
## 4     729 Passeriformes
## 5    5290      Turdidae
## 6 2490714        Turdus
```

From the output we can extract the ID of the encompassing genus, which we will be using later on in the tutorial, since we will to work with data of several species. If your genus only has a single species, maybe pick a different genus.

```
genus_ID = taxon_taxonomy_data$data$genusKey
genus_ID
```

```
## [1] 2490714
```

Alternatively you can also extract the ID of the parent taxon in general, e.g. if you looked up data for a genus you can extract the family ID like this (in the case of my example here it is the ID of the genus since I looked up the taxonomy of a species, which I stored as `taxon_taxonomy_data`):

```
genus_ID = taxon_taxonomy_data$data$parentKey
genus_ID
```

```
## [1] 2490714
```

We can also retrieve a list of popular names (vernacular names) in different languages for our taxon. This list might come in handy if we are to combine the GBIF occurrence data with data from other data-sources, which may not have adopted the same taxonomy. In that case we could search for any matches with this list of vernacular names.

```
taxon_taxonomy_data$names[[1]]$vernacularName
```

```
##  [1] Amsel
##  [2] blackbird
##  [3] merel
##  [4] merle noir
##  [5] blackbird
##  [6] Eurasian Blackbird
##  [7] Merle noir
##  [8] Common Blackbird
##  [9] Eurasian Blackbird
## [10] Eurasian Blackbird / Common Blackbird
## [11] Common Blackbird
## [12] Eurasian Blackbird
## [13] Merle noir
## [14] common blackbird
## [15] Amsel
## [16] common blackbird
## [17] karatavuk
## [18] merel
## [19] merle noir
## [20] mulleja
## [21] mullija
## [22] mullizeza
## [23] mwyalchen
## [24] mëllënja
## [25] qofka e murrme
## [26] qukla
## [27] svarttrost
## [28] Κοινός Κότσυφας
## [29] Кос
## [30] Amsel
## [31] Eurasian blackbird
## [32] blackbird
## [33] Amsel
## [34] Blackbird
## [35] Mirlo Común
## [36] Mustarastas
## [37] Merle noir
## [38] Fekete rigó
## [39] Merlo
## [40] Merel
## [41] Kos
## [42] Melro-preto
## [43] Koltrast
## [44] Solsort
## [45] Amsel
## [46] Common Blackbird
## [47] Kvørkveggja
## [48] Mustarastas
## [49] Solsort
## [50] Svarttrost
## [51] Svartßröstur
## [52] koltrast
## [53] Merel
## [54] Zwarte lijster
```

```
## [55] Common Blackbird
## [56] Eurasian Blackbird
## [57] merle noir
## [58] Amsel
## [59] Common Blackbird
## [60] Eurasian Blackbird
## [61] merle noir
## [62] Amsel
## [63] Common Blackbird
## [64] Merel
## [65] Merle noir
## [66] Merlo
## [67] Mirlo Común
## [68] Solsort
## [69] Svarttrost
## [70] drozd čierny
## [71] fekete rigó
## [72] juodasis strazdas
## [73] koltrast
## [74] kos
## [75] kos (zwyczajny)
## [76] kos černý
## [77] melnais mežastrazds
## [78] melro
## [79] merla
## [80] mustarastas
## [81] musträstas
## [82] Чёрный дрозд
## [83] 乌鸫
## [84] 烏鶇
## 50 Levels: Amsel blackbird merel merle noir ... 烏鶇
```

4. Download occurrence data

Now we have determined our taxon ID and can proceed downloading occurrence data. Downloading occurrence data from GBIF can be done using the `occ_search()` function of the rgbif package. We already touched upon this in yesterday's tutorial, but we'll explore the gbif `occ_search()` function in some more detail today.

Before we start downloading, it is usually a good idea to check first how many occurrences will be downloaded. In general it takes quite a while to download hundreds of thousands of records, while a few thousand up to tens of thousands might be feasible for this exercise. In order to get a count of matches, we only download the metadata of our search (instead of the actual data) by setting `return = 'meta'`. We then call the count value from the exported metadata:

```
occ_search(taxonKey=taxon_id, hasCoordinate = TRUE, return =  "meta")$count
```

```
## [1] 4853745
```

You see in my case there are almost 5 Million records of my target species! If you want to restrict your search by searching for a specific locality/region/country, there are several flags that can be used to restrict the search geographically. Some examples are:

```
locality = c("Gothenburg", "Göteborg")

continent = 'europe'

country = 'SE'
```

In my case I will proceed with donwloading all occurrences for Sweden (country = 'SE'). Feel free to select whatever region or city you're interested in.

```
occ_search(taxonKey=taxon_id, hasCoordinate = TRUE, return =  "meta",country = 'S
E')$count
```

```
## [1] 748102
```

If your count of datapoints is higher than 20,000 you probably want to set a limit to the number of occurrences to be downloaded for the purpose of this exercise, since it will otherwise take a very long time. You can set a download limit in the `occ_search()` function, using the limit=NUM flag, where NUM is the number of records you want to export. In this case I'm setting this number to 5,000. Downloading this many records may take a minute. If your taxon of choice doesn't have that many records, it will donwload as many records as there are and it might thus be finished a lot faster.

```
records <- occ_search(taxonKey=taxon_id, return="data", hasCoordinate=TRUE, limit
=5000,country = 'SE')
```

As a little side note, if you want to download occurrence data for a list of species names (`spp_names`) you can do it like this, by first getting the taxon ids (`keys`) for all your target species, and then feddign this list of ids into the `occ_search()` function. This will return a list of output dataframes, one for each species.

```
spp_names <- c('Hepatica nobilis', 'Anemone nemorosa', 'Taraxacum officinale', 'T
rifolium pratense')
keys <- sapply(spp_names, function(x) name_backbone(name=x, kingdom='plants')$spe
ciesKey, USE.NAMES=FALSE)
spp <- occ_search(taxonKey=keys, limit=100, return='data', country='NO', hasCoord
inate=TRUE) ## return list
```

5. Plot the occurrence data

There is a cool plotting library ggplot2 (https://ggplot2.tidyverse.org/) which allows very easy plotting from dataframes like the one produced from the gbif `occ_search()` command. ggplot2 offers tons of fancy plotting options and is a whole science for itself. If you enjoy making nice plots it's worth checking it out in more detail ((https://ggplot2.tidyverse.org/)). For now we just load the map object and we'll then use it in the plotting command later on.

ggplot2 also has integrated commands to load map data. Using the borders() function, we can very easily load a world map with the following command. This stores the polygon information in a ggplot specific format, which is different to the SpatialPolygons format of the world polygons we were loading yesterday. You can set the color of the map and of the filling of landmasses in this command.

```
library(ggplot2)
map_world <- borders(database = "world", colour = "gray50", fill = "#383838")
```

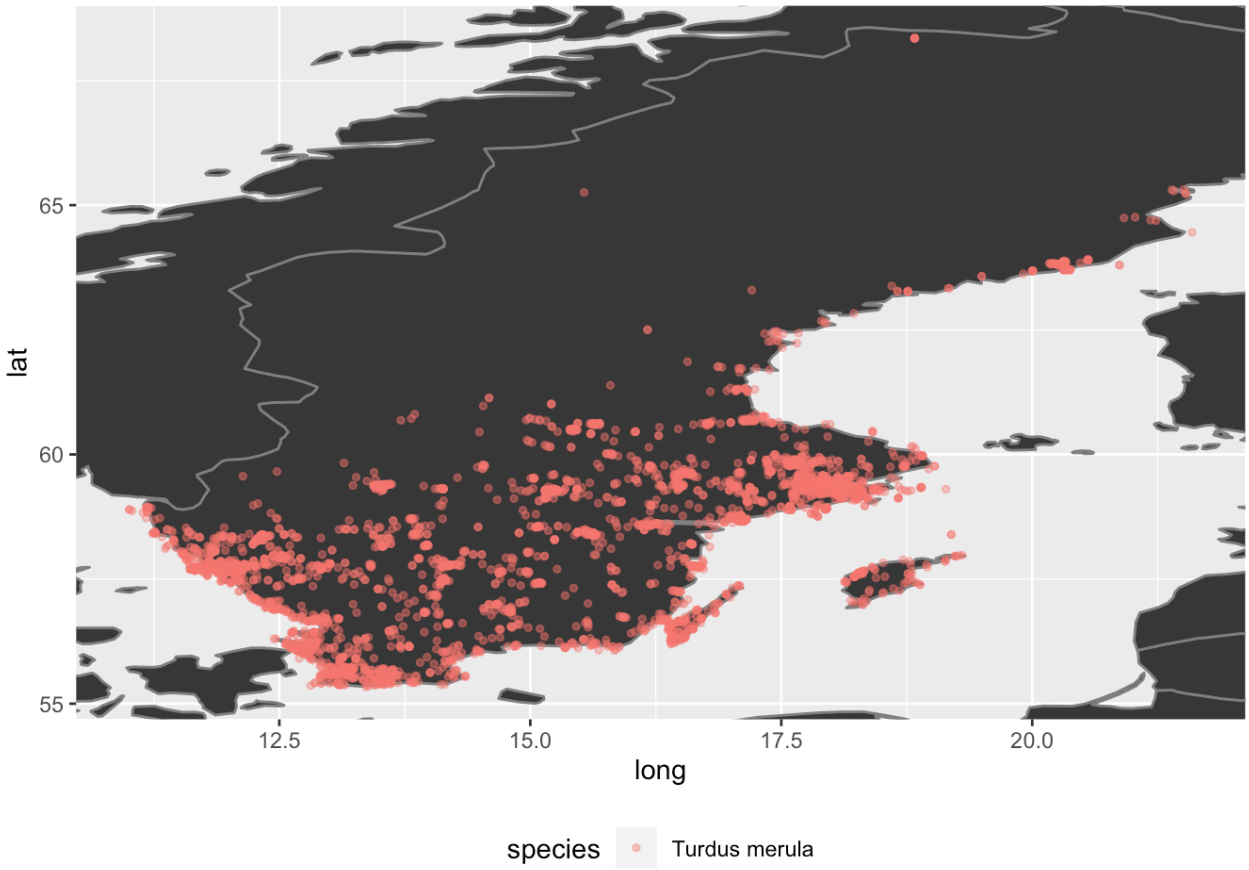
It is usually a good idea to crop the world map to the area where our points are occurring. So let's first define the bounding box, which we can then use to restrict the plotting to only our selected area:

```
xmin<-min(records$decimalLongitude)
xmax<-max(records$decimalLongitude)
ymin<-min(records$decimalLatitude)
ymax<-max(records$decimalLatitude)
```

Now let's plot our points on the map, colored by species name. The logic of the ggplot syntax is accumulative. That means you can add additonal layers/settings to the command by adding lines connnected with a `+` sign.

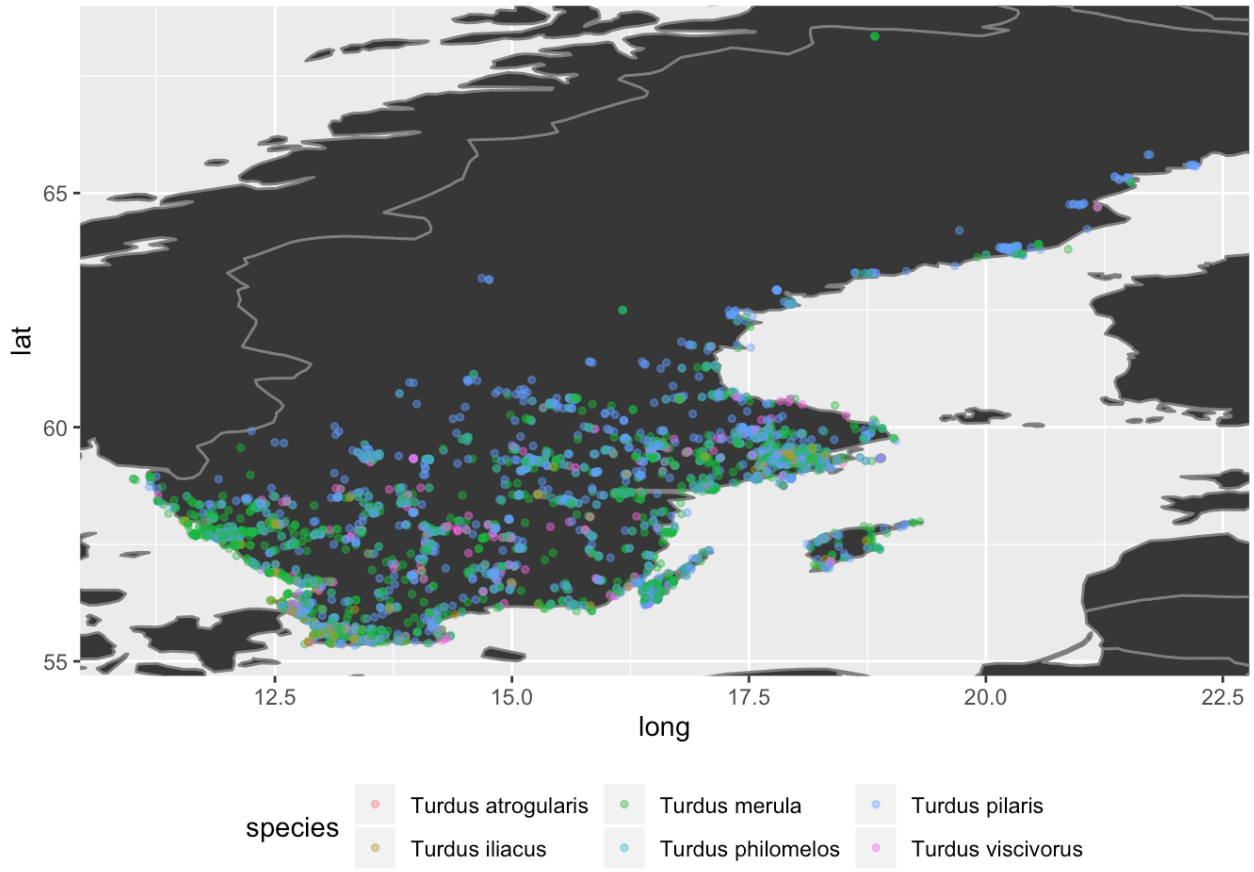
First we just call the function `ggplot()`, then tell it to plot our `map_world` object we created above, then add our gbid dataframe (`records`) using the `geom_point()` function. Within the `geom_point()` commadn we use `aes()` where we provide the column names in the dataframe that correspond to the x-coordinates (`x=`), y-coordinates (`y=`), and the name of the column we want to color our points by (`colour=`), which in this case is by the species column. Then crop the map around the borders (xmin,xmax,ymin,ymax) defined in the previous step (based on the spread of the data), and finally we add a legend using the `theme()` command.

```
ggplot() +
map_world +
geom_point(data = records,  # Specify the data for geom_point()
             aes(x = decimalLongitude,  # Specify the x axis as longitude
                 y = decimalLatitude,
                 colour = species),  # Colour the points based on species name
             alpha = 0.4,  # Set point opacity to 40%
             size = 1) +
coord_cartesian(xlim = c((xmin),(xmax)), ylim = c((ymin),(ymax))) +
theme(legend.position = "bottom")
```



You see that this is somewhat simpler than the way we did it yesterday with loading country or world shape files from separate files and loading them as spatial polygons, transforming the points into SpatialPoints objects, etc. However, it is good to know and understand how to do it the old-school way, but it's also nice to know that there are easier and more user-freindly options out there.

The plot above looks a bit boring since it only contains the data from my one target species. Recreate the plot for your area and target group of choice. Make sure your dataset contains multiple species, e.g. use the `genus_ID` we extracted earlier, to download and plot data for a whole genus. Color the points by the species they belong to.



6. Citing the source data

If you want to be very thorough in your citations, you can export all citations of the data that are present in your downloaded GBIF dataframe (which we stored as `records`). You can easily retrieve this information using the `gbif_citation()` function:

```
gbif_citation(records)
```

```
## [[1]]
## <<rgbif citation>>
##    Citation: Shah M, Coulson S (2020). Artportalen (Swedish Species Observatio
n
##         System). Version 92.192. SLU Artdatabanken. Occurrence dataset
##         https://doi.org/10.15468/kllkyl accessed via GBIF.org on 2020-05-25..
##         Accessed from R via rgbif (https://github.com/ropensci/rgbif) on
##         2020-05-26
##    Rights:
```

In some cases this could be a very long list of references. Alternatively/additionally you can create your own official download request at GBIF, which will assign a DOI to your download that can (and should) be cited when publishing such data. The advantage is that everybody can access the donwload and (hopefully) reproduce your operations on the data. This is the proper way of using GBIF data for publications.

First you need to create a user account at GBIF. This is very simple and fast, just follow this link (https://www.gbif.org/user/profile). Once you created your account and activated it via email, you can execute the lines below in R after replacing the values `USERNAME`, `PASSWORD`, and `EMAIL` with your

account name, passwort, and email address, respectively. You only need to do this once in this session. From here on out the rgbif package will remember your user data, but you'll have to enter them again when you restart R next time.

```
options(gbif_user='USERNAME')
options(gbif_pwd='PASSWORD')
options(gbif_email='EMAIL')
```

Now let's create a download request for all occurrences associated with your chosen taxon. For this you can use the `occ_download()` command (instead of `occ_search()` which we were using before), where you provide the `taxonKey` and specify that you want only records with coordinates (`hasCoordinate`). The `type="and"` setting means that both of these requirements have to be fulfilled (i.e. the `taxonId` needs to match and the records must have coordinates assigned to it).

```
# Get download key
request = occ_download(paste0('taxonKey =',taxon_id),'hasCoordinate = TRUE',type
= "and")
request
```

```
## <<gbif download>>
##   Username: tobiashofmann
##   E-mail: tobiashofmann@gmx.net
##   Download key: 0070686-200221144449610
```

You can get more information about your download request by using the `occ_download_meta()` function or by logging into your gbif user account and checking the download section (https://www.gbif.org/user/download):

```
download_key = occ_download_meta(request[1])
download_key
```

```
## <<gbif download metadata>>
##   Status: RUNNING
##   Format: DWCA
##   Download key: 0070686-200221144449610
##   Created: 2020-05-25T23:43:35.894+0000
##   Modified: 2020-05-25T23:43:36.009+0000
##   Download link: http://api.gbif.org/v1/occurrence/download/request/0070686-20
0221144449610.zip
##   Total records: 0
##   Request:
##     type:  and
##     predicates:
##       > type: equals, key: TAXON_KEY, value: 2490719
##       > type: equals, key: HAS_COORDINATE, value: TRUE
```

It will take 10-20 minutes (depending on the number of records in your query) for this download request to finish compiling. Once the Status: field says `SUCCEEDED` you are ready to retrieve your data. You can do this through R using the `occ_download_get()` function, or you can instead manually download the file by clicking on download on your GBIF webpage. When using the `occ_download_get()` function, provide the path where the zipped folder should be saved (`../output_files/` in my case, replace with your path where you want to save, make sure that the folder exists where you are trying to save it).

```
#key = request[1]
key = "0006240-190415153152247"
occ_download_get(key,'../output_files/')
```

```
## <<gbif downloaded get>>
##   Path: ../output_files//0006240-190415153152247.zip
##   File size: 773.48 MB
```

Extract DOI:

We can now use the official GBIF download we requested to cite our data with a unique DOI identifier. The DOI information can be found on the GBIF download webpage, but is also stored in the output of the `occ_download_meta()` function, which we stored as the variable download_key.

The complete citation of these data should be something along these lines:

```
paste0("GBIF Occurrence Download doi:", download_key[2], " accessed via GBIF.org
on ", Sys.Date())
```

```
## [1] "GBIF Occurrence Download doi:10.15468/dl.d462bd accessed via GBIF.org on
2020-05-26"
```

Load the data:

The main data is stored in a file called `occurrence.txt` in the downloaded zip archive. You can read the data directly form the zip-archive into R, using the unzip function `unz()` together with the `read.table()` function, which reads the data as a dataframe into R. Here we spare our memory by only reading the first 50,000 rows of the data (nrows=50000).

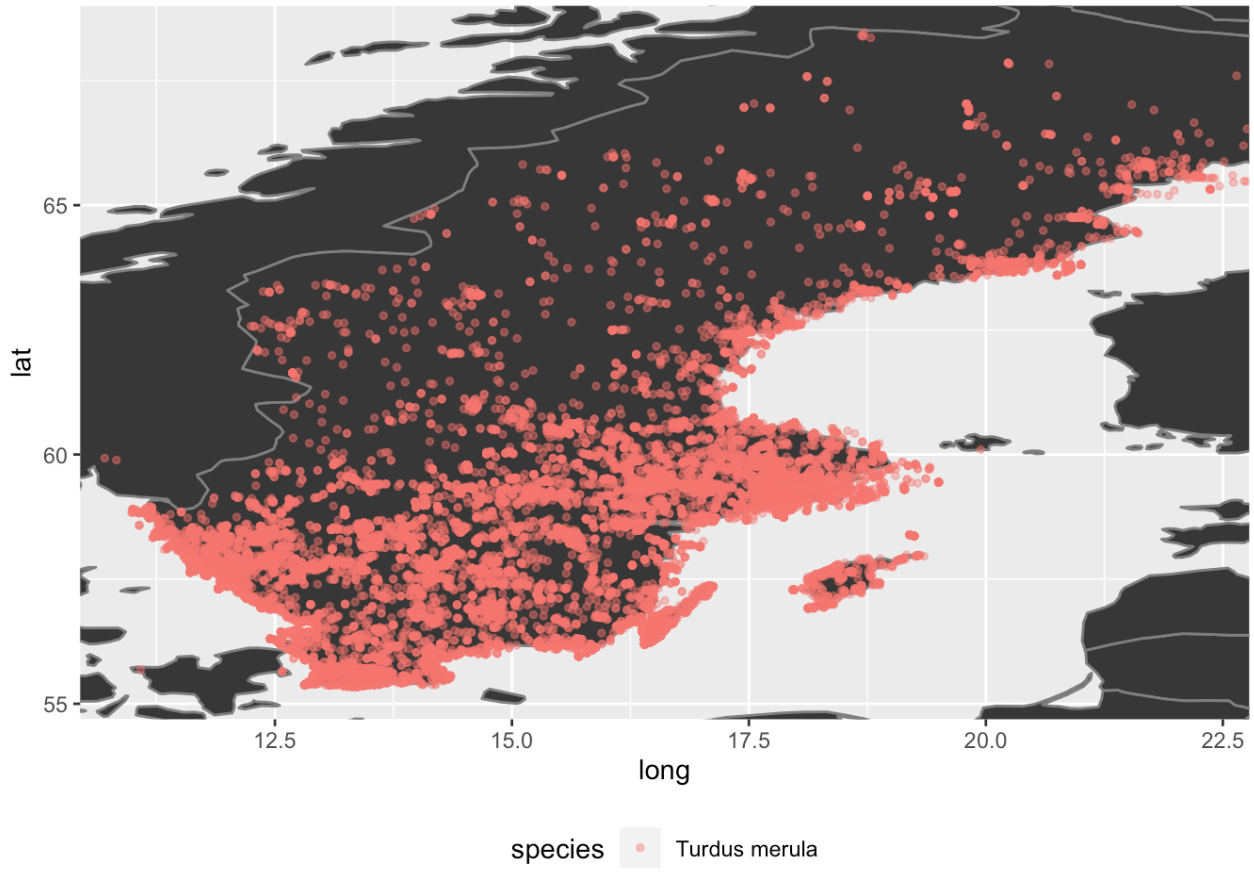
```
doi_data = read.table(unz('../output_files/0006240-190415153152247.zip', "occurre
nce.txt"),quote="\"", nrows=50000, fill = TRUE ,header=T, sep="\t")
```

Plot the data:

One important note before plotting the data is that the downloaded data could contain some strange coordinates that cannot be properly read and will therefore be coded as `NaN`. These coordinates would cause an error in the plotting function and we therefore need to remove them first. The following two lines take care of that (only selecting lines where the condition `!isna()` is fulfilled, i.e. only those lines that are not `NaN`, as the `!` reverses the statement it is followed by):

```
# remove all rows that have NA data in the coordinates
doi_data=doi_data[!is.na(doi_data$decimalLatitude),]
doi_data = doi_data[!is.na(doi_data$decimalLongitude),]
```

After removing the `NaN` coordinates, plot these data in the same manner as we did above with the data directly downloaded from GBIF through the `occ_search()` function. Make sure you understand the difference between the way we downloaded occurrence data with `occ_search()` (dynamic download through R online portal) vs. the way we did it with `occ_download() + occ_download_get()` (API-based DOI-tagged download).



Interactive mapping

As a fun plotting exercise we will plot occurrence data on an interactive OpenStreetView map. The advantage of this is that the user can zoom in and out of the plot and explore large geographic extends in very high detail. Additionally the meta-data (additional values attached to each point, such as species name, etc) of each point can be viewed by clicking on the data point on the map. Let's download a multi species dataset, in this case I'm using a genus ID (genus_ID), which we assigned earlier in this tutorial in order to get multiple species occurrences.

```
genus_records = occ_search(taxonKey=genus_ID, return="data", hasCoordinate=TRUE,
limit=1000,country = 'SE')
```

For plotting interactively we will use the map_leaflet() function as part of the mapr package (The plot might not show, because it can't be displayed by some html viewers. You can view it when downloading this tutorial file in html format and opening it in your html viewer, e.g. Firefox)

```
library(mapr) # rOpenSci r-package for mapping (occurrence data)
map_leaflet(genus_records, lon="decimalLongitude", lat="decimalLatitude", size=5)
```

In the plot you can zoom in and out and you can click on individual points to see which metadata are attached to them.

If you want to play around with different colors, there is a great package called RColorBrewer, which you can use to generate visually pleasing palettes of colors. You can check out the options you can choose from by using the help function `?colorRampPalette()`. Here we first determine the number of species in our data (in case we have several) and then parse it into the function to create a different color for each species.

```
library('RColorBrewer')

n_spp <- length(unique(genus_records$name)) # number of unique taxa in dataframe (USE spp$name, NOT spp$taxonKey)
myColors <- colorRampPalette(brewer.pal(11,"Spectral"))(n_spp) # create color palette with [n_spp] colors
map_leaflet(genus_records, "decimalLongitude", "decimalLatitude", size=5, color=myColors)
```