Species occurrence data (spocc), version 0.1

Introduction

The rOpenSci projects aims to provide programmatic access to scientific data repositories on the web. A vast majority of the packages in our current suite retrieve some form of biodiversity or taxonomic data. Since several of these datasets have been georeferenced, it provides numerous opportunities for visualizing species distributions, building species distribution maps, and for using it analyses such as species distribution models. In an effort to streamline access to these data, we have developed a package called Spocc, which provides a unified API to all the biodiversity sources that we provide. The obvious advantage is that a user can interact with a common API and not worry about the nuances in syntax that differ between packages. As more data sources come online, users can access even more data without significant changes to their code. However, it is important to note that spocc will never replicate the full functionality that exists within specific packages. Therefore users with a strong interest in one of the specific data sources listed below would benefit from familiarising themselves with the inner working of the appropriate packages.

Data Sources

`spocc` currently interfaces with five major biodiversity repositories. Many of these packages have been part of the rOpenSci su

1. Global Biodiversity Information Facility (`rgbif`)
   GBIF is a government funded open data repository with several partner organizations with the express goal of providing access to data on Earth's biodiversity. The data are made available by a network of member nodes, coordinating information from various participant organizations and government agencies.

2. Berkeley Ecoengine (`ecoengine`)
   The ecoengine is an open API built by the Berkeley Initiative for Global Change Biology. The repository provides access to over 3 million specimens from various Berkeley natural history museums. These data span more than a century and provide access to georeferenced specimens, species checklists, photographs, vegetation surveys and resurveys and a variety of measurements from environmental sensors located at reserves across University of California's natural reserve system.

3. **iNaturalist** (`rinat`) iNaturalist provides access to crowd sourced citizen science data on species observations.

4. VertNet (`rvertnet`) Similar to `rgbif`, ecoengine, and `rbison` (see below), VertNet provides access to more than 80 million vertebrate records spanning a large number of institutions and museums primarly covering four major disciplines (mammology, herpetology, ornithology, and icthyology). **Note that we don't currenlty support VertNet data in this package, but we should soon**

5. Biodiversity Information Serving Our Nation (`rbison`)
   Built by the US Geological Survey's core science analytic team, BISON is a portal that provides access to species occurrence data from several participating institutions.

6. eBird (`rebird`)
   ebird is a database developed and maintained by the Cornell Lab of Ornithology and the National Audubon Society. It provides real-time access to checklist data, data on bird abundance and distribution, and communtiy reports from birders.

7. AntWeb (`AntWeb`)
   AntWeb is the world's largest online database of images, specimen records, and natural history information on ants. It is community driven and open to contribution from anyone with specimen records, natural history comments, or images.

Note: It's important to keep in mind that several data providers interface with many of the above mentioned repositories. This means that occurence data obtained from BISON may be duplicates of data that are also available through GBIF. We do not have a way to resolve these duplicates or overlaps at this time but it is an issue we are hoping to resolve in future versions of the package.

Data retrieval

The most significant function in spocc is the `occ` (short for occurrence) function. `occ` takes a query, often a species name, and searches across all data sources specified in the `from` argument. For example, one can search for all occurrences of Sharp-shinned Hawks (*Accipiter striatus*) from the GBIF database with the following R call.

```
library(spocc)
df <- occ(query = "Accipiter striatus", from = "gbif")
```

```
## Loading required package: rjson
```

```
df
```

```
## Summary of results - occurrences found for:
##  gbif  : 25 records across 1 species
##  bison :  0 records across 1 species
##  inat  :  0 records across 1 species
##  ebird :  0 records across 1 species
##  ecoengine :  0 records across 1 species
##  antweb :  0 records across 1 species
```

```
head(df$gbif$data[[1]])
```

```
##                 name       key longitude latitude prov
## 1 Accipiter striatus 768992325    -76.10    4.724 gbif
## 2 Accipiter striatus 773408845    -97.32   32.821 gbif
## 3 Accipiter striatus 773414146   -122.27   37.771 gbif
## 4 Accipiter striatus 859267562   -108.34   36.732 gbif
## 5 Accipiter striatus 859267548   -108.34   36.732 gbif
## 6 Accipiter striatus 859267717   -108.34   36.732 gbif
```

The data returned are part of a `S3` class called `occdat`. This class has slots for the five data sources described above. One can easily switch the source by changing the `from` parameter in the function call above.

Within each data source is the set of species queried. In the above example, we only asked for occurrence data for one species, but we could have asked for any number. Let's say we asked for data for two species: *Accipiter striatus*, and *Pinus contorta*. Then the structure of the response would be

```
response -- |
            | -- gbif ------- |
                              | -- Accipiter_striatus
                              | -- Pinus_contorta

            | -- ecoengine -- |
                              | -- Accipiter_striatus
```

```
                        | -- Pinus_contorta

            | -- inat ------- |
                        | -- Accipiter_striatus
                        | -- Pinus_contorta

            | -- bison ------ |
                        | -- Accipiter_striatus
                        | -- Pinus_contorta

            | -- ebird ------ |
                        | -- Accipiter_striatus
                        | -- Pinus_contorta

            | -- antweb ----- |
                        | -- Accipiter_striatus
                        | -- Pinus_contorta
```

If you only request data from gbif, like `from = 'gbif'`, then the other four source slots are prsent in the response object, but have no data.

You can quickly get just the data by indexing to the data element, like

```
head(df$gbif$data$Accipiter_striatus)
```

```
##                 name       key longitude latitude prov
## 1 Accipiter striatus 768992325    -76.10    4.724 gbif
## 2 Accipiter striatus 773408845    -97.32   32.821 gbif
## 3 Accipiter striatus 773414146   -122.27   37.771 gbif
## 4 Accipiter striatus 859267562   -108.34   36.732 gbif
## 5 Accipiter striatus 859267548   -108.34   36.732 gbif
## 6 Accipiter striatus 859267717   -108.34   36.732 gbif
```

When you get data from multiple providers, the fields returned are slightly different, e.g.:

```
df <- occ(query = "Accipiter striatus", from = c("gbif", "ecoengine"))
head(df$gbif$data$Accipiter_striatus)
```

```
##                 name       key longitude latitude prov
## 1 Accipiter striatus 768992325    -76.10    4.724 gbif
## 2 Accipiter striatus 773408845    -97.32   32.821 gbif
## 3 Accipiter striatus 773414146   -122.27   37.771 gbif
## 4 Accipiter striatus 859267562   -108.34   36.732 gbif
## 5 Accipiter striatus 859267548   -108.34   36.732 gbif
## 6 Accipiter striatus 859267717   -108.34   36.732 gbif
```

```
head(df$ecoengine$data$Accipiter_striatus)
```

```
##                                                                  url
## 1 http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A179318/
## 2  http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A41449/
## 3  http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A64564/
```

```
## 4  http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A12218/
## 5  http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A56349/
## 6  http://ecoengine.berkeley.edu/api/observations/MVZ%3ABird%3A26439/
##   observation_type                        name       country state_province
## 1         specimen   Accipiter striatus velox United States     California
## 2         specimen Accipiter striatus suttoni United States        Arizona
## 3         specimen   Accipiter striatus velox United States     California
## 4         specimen   Accipiter striatus velox United States     California
## 5         specimen   Accipiter striatus velox United States     California
## 6         specimen   Accipiter striatus velox United States     California
##   begin_date   end_date                                          source
## 1 1996-12-05 1996-12-05 http://ecoengine.berkeley.edu/api/sources/1/
## 2 1919-08-26 1919-08-26 http://ecoengine.berkeley.edu/api/sources/1/
## 3 1934-02-10 1934-02-10 http://ecoengine.berkeley.edu/api/sources/1/
## 4 1907-08-23 1907-08-23 http://ecoengine.berkeley.edu/api/sources/1/
## 5 1906-12-16 1906-12-16 http://ecoengine.berkeley.edu/api/sources/1/
## 6 1915-12-31 1915-12-31 http://ecoengine.berkeley.edu/api/sources/1/
##                                     remote_resource geojson.type
## 1 http://arctos.database.museum/guid/MVZ:Bird:179318        Point
## 2  http://arctos.database.museum/guid/MVZ:Bird:41449        Point
## 3  http://arctos.database.museum/guid/MVZ:Bird:64564        Point
## 4  http://arctos.database.museum/guid/MVZ:Bird:12218        Point
## 5  http://arctos.database.museum/guid/MVZ:Bird:56349        Point
## 6  http://arctos.database.museum/guid/MVZ:Bird:26439        Point
##   longitude latitude      prov
## 1    -122.1    37.87 ecoengine
## 2    -109.4    31.93 ecoengine
## 3    -122.3    37.90 ecoengine
## 4    -116.9    34.18 ecoengine
## 5    -122.2    37.37 ecoengine
## 6    -114.7    33.43 ecoengine
```

We provide a function `occ2df` that pulls out a few key columns needed for making maps:

```
head(occ2df(df))
```

```
##                 name longitude latitude prov
## 1 Accipiter striatus    -76.10    4.724 gbif
## 2 Accipiter striatus    -97.32   32.821 gbif
## 3 Accipiter striatus   -122.27   37.771 gbif
## 4 Accipiter striatus   -108.34   36.732 gbif
## 5 Accipiter striatus   -108.34   36.732 gbif
## 6 Accipiter striatus   -108.34   36.732 gbif
```

Fix names

One problem you often run in to is that there can be various names for the same taxon in any one source. For example:

```
df <- occ(query = "Pinus contorta", from = c("gbif", "inat"), limit = 50)
head(df$gbif$data$Pinus_contorta[, 1:2])
```

```
##                               name       key
## 1 Pinus contorta Douglas ex Loudon 856965570
## 2 Pinus contorta Douglas ex Loudon 856964858
## 3                   Pinus contorta 773428981
## 4                   Pinus contorta 866502616
## 5                   Pinus contorta 866517786
## 6                   Pinus contorta 856022134
```

```
head(df$inat$data$Pinus_contorta[, 1:2])
```

```
##                        name                  Datetime
## 1           Pinus contorta 2014-02-22 00:00:00 +0000
## 2  Pinus contorta contorta 2014-01-17 00:00:00 +0000
## 3           Pinus contorta 2013-12-23 10:55:03 +0000
## 4           Pinus contorta 2013-12-23 11:02:55 +0000
## 5 Elaphocordyceps capitata 2013-12-11 14:05:22 +0000
## 6 Pinus contorta murrayana 2013-10-01 00:00:00 +0000
```

This is fine, but when trying to make a map in which points are colored for each taxon, you can have many colors for a single taxon, where instead one color per taxon is more appropriate. There is a function in `spocc` called `fixnames`, which has a few options in which you can take the shortest names (usually just the plain binomials like *Homo sapiens*), or the original name queried, or a vector of names supplied by the user.

```
df <- fixnames(df, how = "shortest")
head(df$gbif$data$Pinus_contorta[, 1:2])
```

```
##             name       key
## 1 Pinus contorta 856965570
## 2 Pinus contorta 856964858
## 3 Pinus contorta 773428981
## 4 Pinus contorta 866502616
## 5 Pinus contorta 866517786
## 6 Pinus contorta 856022134
```

```
head(df$inat$data$Pinus_contorta[, 1:2])
```

```
##             name                  Datetime
## 1 Pinus contorta 2014-02-22 00:00:00 +0000
## 2 Pinus contorta 2014-01-17 00:00:00 +0000
## 3 Pinus contorta 2013-12-23 10:55:03 +0000
## 4 Pinus contorta 2013-12-23 11:02:55 +0000
## 5 Pinus contorta 2013-12-11 14:05:22 +0000
## 6 Pinus contorta 2013-10-01 00:00:00 +0000
```

```
df_comb <- occ2df(df)
head(df_comb)
```

```
##             name longitude latitude prov
## 1 Pinus contorta    -3.630    55.73 gbif
## 2 Pinus contorta    -3.644    55.72 gbif
## 3 Pinus contorta  -120.040    38.87 gbif
```

```
## 4 Pinus contorta  -122.271    47.78 gbif
## 5 Pinus contorta  -123.803    39.29 gbif
## 6 Pinus contorta    20.350    63.71 gbif
```

```
tail(df_comb)
```

```
##               name longitude latitude prov
## 95  Pinus contorta    -120.2    39.43 inat
## 96  Pinus contorta    -124.2    47.03 inat
## 97  Pinus contorta    -124.2    47.03 inat
## 98  Pinus contorta        NA       NA inat
## 99  Pinus contorta    -116.3    44.77 inat
## 100 Pinus contorta    -124.2    47.03 inat
```

Visualization routines

Interactive maps

Leaflet.js

Leaflet JS is an open source mapping library that can leverage various layers from multiple sources. Using the `leafletR` library, it's possible to generate a local `geoJSON` file and a html file of species distribution maps. The folder can easily be moved to a web server and served widely without any additional coding.

It's also possible to render similar maps with Mapbox by committing just the geoJSON file to GitHub or posting it as a gist on GitHub. All the remaining fields will become part of a table inside a tooltip, providing a extremely quick and easy way to serve up interactive maps. This is especially useful when users do not have their own web hosting options.

Here is an example of making a leaflet map:

```
spp <- c("Danaus plexippus", "Accipiter striatus", "Pinus contorta")
dat <- occ(query = spp, from = "gbif", gbifopts = list(georeferenced = TRUE))
data <- occ2df(dat)
mapleaflet(data = data, dest = ".")
```

Geojson map as a Github gist

You can also create interactive maps via the `mapgist` function. You have to have a Github account to use this function. Github accounts are free though, and great for versioning and collaborating on code or

papers. When you run the `mapgist` function it will ask for your Github username and password. You can alternatively store those in your `.Rprofile` file by adding entries for username (`options(github.username = 'username')`) and password (`options(github.password = 'password')`).

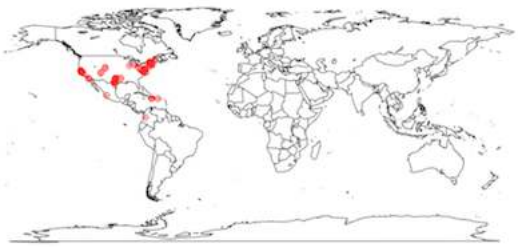
```
spp <- c("Danaus plexippus", "Accipiter striatus", "Pinus contorta")
dat <- occ(query = spp, from = "gbif", gbifopts = list(georeferenced = TRUE))
dat <- fixnames(dat)
dat <- occ2df(dat)
mapgist(data = dat, color = c("#976AAE", "#6B944D", "#BD5945"))
```

Static maps

base plots

Base plots, or the built in plotting facility in R accessed via `plot()`, is quite fast, but not easy or efficient to use, but are good for a quick glance at some data.

```
spnames <- c("Accipiter striatus", "Setophaga caerulescens", "Spinus tristis")
out <- occ(query = spnames, from = "gbif", gbifopts = list(georeferenced = TRUE))
plot(out, cex = 1, pch = 10)
```



ggplot2

`ggplot2` is a powerful package for making visualizations in R. Read more about it here. We created a simple wrapper function `mapggplot` to make a ggplot2 map from occurrence data using the `ggmap` package, which is built on top of `ggplot2`. Here's an example:

```
ecoengine_data <- occ(query = "Lynx rufus californicus", from = "ecoengine")
mapggplot(ecoengine_data)
```

Upcoming features

- As soon as we have an updated `rvertnet` package, we'll add the ability to query VertNet data from `spocc`.
- We will add `rCharts` as an official import once the package is on CRAN (Eta end of March)
- We're helping on a new package `rMaps` to make interactive maps using various Javascript mapping libraries, which will give access to a variety of awesome interactive maps. We will integrate `rMaps` once it's on CRAN.
- We'll add a function to make interactive maps using RStudio's Shiny in a future version.