

R Python, and Ruby clients for GBIF species occurrence data

Scott Chamberlain^{*,a}, Carl Boettiger^b

^a*rOpenSci, Museum of Paleontology, University of California, Berkeley, CA, USA*

^b*rOpenSci, Department of Environmental Science, Policy and Management, University of California, Berkeley, CA, USA*

Abstract

Corresponding Author:

Scott Chamberlain

rOpenSci, Museum of Paleontology, University of California, Berkeley, CA, USA

Email address: scott@ropensci.org

*Corresponding author

Email addresses: [scott\(at\)ropensci.org](mailto:scott@ropensci.org) (Scott Chamberlain), [carl\(at\)ropensci.org](mailto:carl@ropensci.org) (Carl Boettiger)

10 Background. The number of individuals of each species in a given location forms the basis for many
11 sub-fields of ecology and evolution. Data on individuals, including which species, and where they're
12 found can be used for a large number of research questions. Global Biodiversity Information Facility
13 (hereafter, GBIF) is the largest of these. Programmatic clients for GBIF would make research dealing
14 with GBIF data much easier and more reproducible.

15 Methods. We have developed clients to access GBIF data for each of the R, Python, and Ruby
16 programming languages: `rgbif`, `pygbif`, `gbifrb`.

17 Results. For all clients we describe their design and utility, and demonstrate some use cases.

Discussion. Programmatic access to GBIF will facilitate more open and reproducible science - the three
GBIF clients described herein are a significant contribution towards this goal.

18 Introduction

19 Perhaps the most fundamental element in many fields of ecology is the individual organism. The number
20 of individuals of each species in a given location forms the basis for many sub-fields of ecology and
21 evolution. Some research questions necessitate collecting new data, while others can easily take advantage
22 of existing data. In fact, some ecology fields are built largely on existing data, e.g., macro-ecology
23 (Brown, 1995; Beck et al., 2012).

24 Data on individuals, including which species, and where they're found, can be used for a large number of
25 research questions. Biodiversity records have been used for a suite of other use cases: validating habitat
26 suitability models with real occurrence data (Ficetola et al., 2014); ancestral range reconstruction
27 (Ferretti et al., 2015; María Mendoza et al., 2015); development of invasive species watch lists (Faulkner
28 et al., 2014); evaluating risk of invasive species spread (Febbraro et al., 2013); and effects of climate
29 change on future biodiversity (Brown et al., 2015).

30 In addition to wide utility, this data is important for conservation. Biodiversity loss is one of the greatest
31 challenges of our time (Pimm et al., 2014), and some have called this the sixth great mass extinction
32 (Ceballos et al., 2015). Given this challenge there is a great need for data on specimen records, whether
33 collected from live sightings in the field or specimens in museums.

34 Global Biodiversity Information Facility

35 There are many online services that collect and maintain specimen records. However, Global Biodiversity
36 Information Facility (hereafter, GBIF, <http://www.gbif.org>) is the largest collection of biodiversity
37 records globally, currently with 820 million records, roughly 5.9 million taxa, 36,000 datasets from
38 1,300 publishers (as of 2016-02-09). Many large biodiversity warehouses such as iNaturalist (<http://www.inaturalist.org>), VertNet (<http://vertnet.org>), and USGS's Biodiversity Information Serving
39 Our Nation (BISON; <http://bison.usgs.ornl.gov>) all feed into GBIF.

41 The most important organizational level in GBIF occurrence data is the occurrence record. The
42 fields in a record vary, but include information about taxonomy (kingdom, phylum, genus, species
43 names) and their identifiers, dataset metadata, and locality information including geospatial position.
44 Going upstream, each record is part of a dataset, where each dataset is submitted by an organization,
45 organizations are organized into nodes, datasets are published through institutions (which may be
46 hosted at another organization), and a network is a group of datasets (managed by GBIF).

Each occurrence record has some taxonomic name associated with it, which itself is linked to a lot of other taxonomic data - including a master taxonomic backbone that integrates taxonomies across many taxonomic authorities.

The organization of GBIF matters because you can navigate GBIF data through these hierarchical organizational levels - it helps to be familiar with the terminology and how each group relates to another.

The clients

Although we discuss libraries for R, Python, and Ruby here, we focus mostly on the R library `rgbif` as it has seen the most developer and user attention, and is the most mature.

rgbif

Herein, we describe the `rgbif` software package (Chamberlain et al.) for working with GBIF data in the R programming environment (R Core Team, 2014). R is a widely used language in academia, as well as non-profit and private sectors. Importantly, R makes it easy to execute all steps of the research process, including data management, data manipulation and cleaning, statistics, and visualization. Thus, an R client for getting GBIF data is a powerful tool to facilitate reproducible research.

The `rgbif` package is nearly completely written in R (a small Javascript library is included for reading well known text (Herring, 2011)), uses an [MIT license](#) to maximize use everywhere. `rgbif` is developed publicly on GitHub at <https://github.com/ropensci/rgbif>, where development versions of the package can be installed, and bugs and feature requests reported. Stable versions of `rgbif` can be installed from [CRAN](#), the distribution network for R packages. `rgbif` is part of the rOpenSci project (<http://ropensci.org>), a developer network making R software to facilitate reproducible research.

pygbif

`pygbif` (Chamberlain) is a Python library for working with GBIF data in the Python programming environment. Python is a general purpose programming language used widely in all sectors, and for all parts of software development including server and client side use cases. Python is used exclusively in some scientific disciplines (e.g., astronomy), and has partial usage in other disciplines. A Python client for GBIF data is an important tool given the even wider usage of Python than R, though maybe slightly less than R for ecology/biology disciplines.

```
pip install pygbif
```

```
import pygbif
```

74 The `pygbif` library is less mature and complete than the R package. It also uses an [MIT license](#) to
75 maximize use everywhere. `pygbif` is developed publicly on GitHub at <https://github.com/sckott/pygbif>,
76 where development versions of the package can be installed, and bugs and feature requests reported.
77 Stable versions of `pygbif` can be installed from [pypi](#), the distribution network for Python libraries.

78 *gbifrb*

79 `gbifrb` (Chamberlain) is a library for working with GBIF data in the Ruby programming environment.
80 Like Python, Ruby is a general purpose programming language used widely in all sectors. Unlike
81 Python, Ruby is not used extensively in scientific disciplines. However, a Ruby client for GBIF data
82 can be an important tool given how widely Ruby is used for web and web service development.

```
gem install gbifrb
```

```
require 'gbifrb'
```

83 The `gbifrb` library is less mature and complete than the R and Python libraries. It also uses
84 an [MIT license](#) to maximize use everywhere. `gbifrb` is developed publicly on GitHub at <https://github.com/sckott/gbifrb>, where development versions of the package can be installed, and bugs and
85 feature requests reported. Stable versions of `gbifrb` can be installed from [Rubygems][gemgbif], the
86 distribution network for Ruby libraries.
87

88 *Library interfaces*

89 `rgbif`, `pygbif`, and `gbifrb` are designed following the [GBIF Application Programming Interface](#), or
90 API. The GBIF API has four major components: registry, taxonomic names, occurrences, and maps. We
91 also include functions to interface with the OAI-PMH GBIF service; only dataset (registry) information
92 is available via this service, however. An interface to the GBIF maps API is in development for `rgbif`,
93 but is non-existent for both `pygbif` and `gbifrb`. All three libraries have a suite of functions dealing
94 with each of registry, taxonomic, names, and occurrences - we'll go through each in turn describing
95 design of the user interface and example usage.

96 *GBIF headers*

97 With each request `rgbif`, `pygbif`, `gbifrb` make to GBIF's API, we send request headers that tell GBIF
98 what library the request is coming from, including what version of the library. This helps GBIF know
99 what proportion of requests are coming from which library, and therefore from R vs. Python vs. Ruby;
100 this information is helpful for GBIF in thinking about how people are using GBIF data.

101 *Registry*

102 The GBIF registry API services are spread across five sets of functions via the main GBIF API:

- 103 • Datasets
- 104 • Installations
- 105 • Networks
- 106 • Nodes
- 107 • Organizations

108 Dataset information in general is available via the OAI-PMH service, functions in `rgbif` prefixed with
109 `gbif_oai_`, but not available in `pygbif` or `gbifrb` yet.

110 Datasets are owned by organizations. Organizations are endorsed by nodes to share datasets with GBIF.
111 Datasets are published through institutions, which may be hosted at another organization. A network
112 is a group of datasets (managed by GBIF). Datasets are the units that matter the most with respect
113 to registry information, while installations, networks, nodes, and organizations are simply higher level
114 organizational structure.

115 *Datasets*

116 Dataset functions include search, dataset metadata retrieval, and dataset metrics. Searching for datasets
117 is an important part of the discovery process. One can search for datasets on the GBIF web portal.
118 However, programmatic searching using any of these libraries is more powerful. Identifying datasets
119 appropriate for a research question is helpful as you can get metadata for each dataset, and track down
120 dataset specific problems, if any.

121 The `dataset_search()` function in `rgbif` is one way to search for datasets. Here, we search for the
122 term “oregon”, which finds any datasets that have words matching that term.

```
res <- dataset_search(query = "oregon")
res$data$datasetTitle[1:10]
#> [1] "Oregon State Ichthyology Collection"
#> [2] "Oregon State University Herpetological Collection"
#> [3] "Mygalomorph spiders from southwestern Oregon, USA, with descriptions of four new species"
#> [4] "A new species of Helobdella (Hirudinida: Glossiphoniidae) from Oregon, USA"
#> [5] "Annotated Checklist of the large branchiopod crustaceans of Idaho, Oregon and Washington"
#> [6] "A new species of Chrysobothris Eschscholtz from Oregon and Washington, with notes on other species"
#> [7] "Three new species of Grylloblatta Walker (Insecta: Grylloblattodea: Grylloblattidae), from Oregon"
#> [8] "A new species of Cladotanytarsus (Lenziella) from Oregon supports the systematic concept of the genus"
#> [9] "A new monster from southwest Oregon forests: Cryptomasterbehemoth sp. n. (Opiliones, Laniatores)"
#> [10] "Two new species of Fluminicola (Caenogastropoda, Lithoglyphidae) from southwest Oregon, USA"
```

123 See also `datasets()` and `dataset_suggest()` in `rgbif` for searching for datasets.

124 In Python, we can similarly search for datasets. Here, search for datasets of type `OCCURRENCE`:

```
from pygbif import registry
registry.datasets(type="OCCURRENCE")
```

125 In Ruby, we can do the same. Here, search for datasets of type `OCCURRENCE`:

```
require 'gbifrb'
registry = Gbif::Registry
registry.datasets(type: "OCCURRENCE")
```

126 *Dataset metrics.* Dataset metrics are another useful way of figuring out what datasets you may want to
127 use. One drawback is that these metrics data are only available for datasets of type *checklist*, but there
128 are quite a lot of them (21697).

129 Here, in R we search for dataset metrics for a single dataset, with uuid `ec93a739-1681-4b04-b62f-3a687127a17f`,
130 a checklist of the ants (Hymenoptera: Formicidae) of the World.

```
res <- dataset_metrics(uuid='ec93a739-1681-4b04-b62f-3a687127a17f')
data.frame(rank = names(res$countByRank),
           count = unname(unlist(res$countByRank)))
```

rank	count
SPECIES	13710
SUBSPECIES	3234
GENUS	726
TRIBE	53
SUBFAMILY	20
FAMILY	2
KINGDOM	1
PHYLUM	1
CLASS	1
ORDER	1

131 And in Python, get metrics for the same dataset as above:

```
from pygbif import registry
registry.dataset_metrics(uuid='ec93a739-1681-4b04-b62f-3a687127a17f')
```

132 The same in Ruby:

```
require 'gbifrb'
registry = Gbif::Registry
registry.dataset_metrics(uuid: 'ec93a739-1681-4b04-b62f-3a687127a17f')
```

133 *Networks, nodes, and installations*

134 Networks, nodes and installations are at a higher level of organization above datasets, but can be
 135 useful if you want to explore data from given organizations. Here, in R we search for the first 10 GBIF
 136 networks, returning just the title field.


```

networks(limit = 10)$data$title
#> [1] "GBIF Backbone Sources"
#> [2] "Canadensys"
#> [3] "Southwest Collections of Arthropods Network (SCAN)"
#> [4] "VertNet"
#> [5] "Dryad"
#> [6] "GBIF Network"
#> [7] "The Knowledge Network for Biocomplexity (KNB) "
#> [8] "Online Zoological Collections of Australian Museums (OZCAM)"
#> [9] "Catalogue of Life"
#> [10] "Ocean Biogeographic Information System (OBIS)"

```

137 And in Python:

```

from pygbif import registry
registry.networks(limit = 10)

```

138 And in Ruby:

```

require 'gbifrb'
registry = Gbif::Registry
registry.networks(limit: 10)

```

139 *Taxonomic names*

140 The GBIF taxonomic names API services are spread across five functions in `rgbif`:

- 141 • Search GBIF name backbone - `name_backbone()`
- 142 • Search across all checklists - `name_lookup()`
- 143 • Quick name lookup - `name_suggest()`
- 144 • Name usage of a name according to a checklist - `name_usage()`
- 145 • GBIF name parser - `parsenames()`

146 pygbif and gbifrb have all the same functions, except the name parser goes by `name_parser()` in
147 pygbif and gbifrb.

148 The goal of these name functions is often to settle on a taxonomic name known to GBIF's database.
149 This serves two purposes: 1) when referring to a taxonomic name, you can point to a URI on the
150 Internet, and 2) you can search for metadata on a taxon, and occurrences of that taxon in GBIF.

151 Taxonomic names are particularly tricky. Many different organizations have their own unique codes for
152 the same taxonomic names, and some taxonomic groups have preferred sources for the definitive names
153 for that group. That's why it's best to determine what name GBIF uses, and its associated identifier,
154 for the taxon of interest instead of simply searching for occurrences with a taxonomic name.

155 When searching for occurrences (see below) you can search by taxonomic name (and other filters, e.g.,
156 taxonomic rank), but you're probably better off figuring out the taxonomic key in the GBIF backbone
157 taxonomy, and using that to search for occurrences. The `taxonkey` parameter in the GBIF occurrences
158 API expects a GBIF backbone taxon key.

159 *GBIF Backbone*

160 The GBIF backbone taxonomy is used in GBIF to have a consistent way to refer to taxonomic
161 names throughout their services. The backbone has 5869207 unique names and 2818534 species
162 names. The backbone taxonomy is also a dataset with key `d7dddbf4-2cf0-4f39-9b2a-bb099caae36c`
163 (<https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c>).

164 We can search the backbone taxonomy with the function `name_backbone()` in all three clients. Here,
165 we're searching for the name *Poa*, restricting to genera, and the family *Poaceae*, in R

```
res <- name_backbone(name='Poa', rank='genus', family='Poaceae')
res[c('usageKey', 'kingdom')]
#> $usageKey
#> [1] 2704173
#>
#> $kingdom
#> [1] "Plantae"
```

166 And in Python

```

from pygbif import species
res = species.name_backbone(name='Poa', rank='genus', family='Poaceae')
[ res[x] for x in ['usageKey', 'kingdom'] ]

```

167 And in Ruby

```

require 'gbifrb'
species = Gbif::Species
res = species.name_backbone(name: 'Poa', rank: 'genus', family: 'Poaceae')
res.select { |k,v| k.match(/usageKey|kingdom/) }

```

168 *Name searching*

169 One of the quickest ways to search for names is using `name_suggest()`, which does a very quick search
 170 and returns minimal data. Here, we're searching for the query term *Pum*, and we get back many names:

```

name_suggest(q='Pum', limit = 6)

```

	key	canonicalName	rank
2142856	Althepus	pum	SPECIES
8589398	Pumiliopimoidae		FAMILY
8783253	Pumililema		GENUS
4823360	Pumiliopareia		GENUS
4635949	Pumilina		GENUS
4648228	Pumilopaurus		GENUS

171 The same in Python

```

from pygbif import species
species.name_suggest(q='Pum', limit = 6)

```

172 And in Ruby

```
require 'gbifrb'
species = Gbif::Species
species.name_suggest(q: 'Pum', limit: 6)
```

173 With these results, you can then proceed to search for occurrences with the taxon key(s), or drill down
174 further with other name searching functions to get the exact taxon of interest.

175 *Occurrences*

176 GBIF provides two ways to get occurrence data: through the `/occurrence/search` route (see
177 `occ_search` in `rgbif`, `occurrences.search` in `pygbif`, `Occurrences.search` in `gbifrb`), or via the
178 `/occurrence/download` route (many functions, see below).

179 `occ_search()/occurrences.search/Occurrences.search` are the main functions for the search route,
180 and are more appropriate when you want less data, while the download functions are more appropriate
181 for larger data requests.

182 Small vs. large amounts of data of course is all relative. GBIF imposes for any given search a limit of
183 200,000 records in the search service, after which point you can't download any more records for that
184 search. However, you can download more records for different searches.

185 We think the search service is still quite useful for many people even given the 200,000 limit. For those
186 that need more data, we have created a similar interface in the download functions that should be easy
187 to use with minimal work. Users should take note that using the download service has a few extra steps
188 to get data into R, but is straight-forward.

189 The download service, like the occurrence search service, is rate-limited. That is, you can only have
190 one to three downloads running simultaneously for your user credentials. However, simply check when
191 a download job is complete, then you can start a new download request. See “Queuing Download
192 Requests” below for help automating many download requests in R.

193 *Download API*

194 The download API syntax is similar to the occurrence search API in that the same parameters are
195 used, but the way in which the query is defined is different. For example, in the download API you can
196 do greater than searches (i.e., `latitude > 50`), whereas you cannot do that in the occurrence search

197 API. Thus, unfortunately, we couldn't make the query interface exactly the same for both search and
198 download functions.

199 Using the download service can consist of as few as three steps: 1) Request data via a search; 2)
200 Download data; 3) Import data into R.

201 Request data download given a query. Here, we search for the taxon key 3119195, which is the key for
202 *Helianthus annuus* (<http://www.gbif.org/species/3119195>).

```
occ_download('taxonKey = 3119195')  
#> <<gbif download>>  
#> Username: xxxx  
#> E-mail: xxxx  
#> Download key: 0000840-150615163101818
```

203 You can check on when the download is ready using the functions `occ_download_list()` and
204 `occ_download_meta()`. When it's ready use `occ_download_get()` to download the dataset to your
205 computer.

```
(res <- occ_download_get("0000840-150615163101818", overwrite = TRUE))  
#> <<gbif downloaded get>>  
#> Path: ./0000840-150615163101818.zip  
#> File size: 3.19 MB
```

206 What's printed out above is a very brief summary of what was downloaded, the path to the file, and its
207 size (in human readable form).

208 Next, read the data in to R using the function `occ_download_import()`.

```
library("dplyr")  
dat <- occ_download_import(res)  
dat %>%  
  select(gbifID, decimalLatitude, decimalLongitude)  
#>      gbifID abstract accessRights accrualMethod accrualPeriodicity accrualPolicy alternative  
#> 1  725767384      NA              NA              NA              NA              NA
```

```

#> 2 725767447 NA NA NA NA NA
#> 3 725767450 NA NA NA NA NA
#> 4 725767513 NA NA NA NA NA
#> 5 725767546 NA NA NA NA NA
#> 6 725767579 NA NA NA NA NA
#> 7 725767609 NA NA NA NA NA
#> 8 725767645 NA NA NA NA NA
#> 9 725767678 NA NA NA NA NA
#> 10 725767681 NA NA NA NA NA
#> .. ... .. ... .. ... .. ...
#> Variables not shown: available (lgl), bibliographicCitation (chr), conformsTo (lgl), contribu
#> coverage (lgl), created (chr), creator (lgl), date (lgl), dateAccepted (lgl), dateCopyri
#> (lgl), dateSubmitted (lgl), description (lgl), educationLevel (lgl), extent (lgl), forma
#> hasFormat (lgl), hasPart (lgl), hasVersion (lgl), identifier (chr), instructionalMethod

```

209 In Python

```

from pygbif import occurrences as occ
occ.download('taxonKey = 3119195')
(res = occ.download_get("0000840-150615163101818", overwrite = True))

```

210 We don't have pygbif functionality at the moment for importing data, but it's coming soon.

211 The Ruby library gbifrb does not yet have occurrence download functionality.

212 *Downloaded data format.* The downloaded dataset from GBIF is a Darwin Core Archive (DwC-A), an
 213 internationally recognized biodiversity informatics standard (<http://rs.tdwg.org/dwc/>). The DwC-A
 214 downloaded is a compressed folder with a number of files, including metadata, citations for each of the
 215 datasets included in the download, and the data itself, in separate files for each dataset as well as one
 216 single .txt file. In `rgbif::occ_download_import()`, we simply fetch data from the .txt file. If you
 217 want to dig into the metadata, citations, etc., it is easily accessible from the folder on your computer.

218 *Search API*

219 The search API follows the GBIF API and is broken down into the following functions:

- 220 • Get a single numeric count of occurrences - rgbif: `occ_count()` / pygbif: `occurrences.count`
221 / gbifrb: `Occurrences.count`
- 222 • Search for occurrences - rgbif: `occ_search()` / pygbif: `occurrences.search` / gbifrb:
223 `Occurrences.search`
- 224 • A simplified and optimized version of rgbif: `occ_search()` or `occ_data()` / none / none
- 225 • Get occurrences by occurrence identifier - rgbif: `occ_get()` / pygbif: `occurrences.get` /
226 `gbifrb: Occurrences.get`
- 227 • Get occurrence metadata - rgbif: `occ_metadata()` / pygbif: `various` / gbifrb: `various`

228 *Search for occurrences.* The main search work-horse is `occ_search()`. This function allows very flexible
229 search definitions. In addition, this function does paging internally, making it such that the user does
230 not have worry about the 300 records per request limit - but of course we can't go over the 200,000
231 maximum limit.

232 The output of `occ_search()` presents a compact `data.frame` so that no matter how large the
233 `data.frame`, the output is easily assessed because only a few of the records (rows) are shown, only a few
234 columns are shown (with others shown in name only), and metadata is shown on top of the `data.frame`
235 to indicate data found and returned, media records found, unique taxonomic hierarchies returned, and
236 the query executed.

237 The output of these examples, except one, aren't shown.

238 Search by species name, using `name_backbone()` first to get key

239 **R**

```
library(rgbif)

(key <- name_suggest(q = 'Helianthus annuus', rank = 'species'))$key[1])
#> [1] 9206251

occ_search(taxonKey = key, limit = 2)
#> Records found [17858]
#> Records returned [2]
#> No. unique hierarchies [1]
#> No. media records [1]
```

```
#> No. facets [0]
#> Args [limit=2, offset=0, taxonKey=9206251, fields=all]
#> # A tibble: 2 x 75
#>           name          key decimalLatitude decimalLongitude
#>           <chr>        <int>          <dbl>          <dbl>
#> 1 Helianthus annuus 1433793045      59.66859      16.54257
#> 2 Helianthus annuus 1434024463      63.71622      20.31247
#> # ... with 71 more variables: issues <chr>, datasetKey <chr>,
#> #   publishingOrgKey <chr>, publishingCountry <chr>, protocol <chr>,
#> #   lastCrawled <chr>, lastParsed <chr>, crawlId <int>, extensions <chr>,
#> #   basisOfRecord <chr>, ...
```

240 Python

```
from pygbif import species
from pygbif import occurrences as occ
key = species.name_suggest(q = 'Helianthus annuus', rank = 'species')['data'][0]['key']
occ.search(taxonKey = key, limit = 2)
```

241 Ruby

```
require 'gbifrb'
species = Gbif::Species
occ = Gbif::Occurrences
key = species.name_suggest(q: 'Helianthus annuus', rank: 'species')['data'][0]['key']
occ.search(taxonKey: key, limit: 2)
```

242 Instead of getting a taxon key first, you can search for a name directly

243 R

```
occ_search(scientificName = 'Ursus americanus')
```

244 Python


```
occ.search(scientificName = 'Ursus americanus')
```

245 Ruby

```
occ.search(scientificName: 'Ursus americanus')
```

246 Search for many species

247 R

```
splist <- c('Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa')
keys <- sapply(splist, function(x) name_suggest(x)$key[1], USE.NAMES = FALSE)
occ_search(taxonKey = keys, limit = 5, return = 'data')
```

248 Python

```
from pygbif import species
from pygbif import occurrences as occ

splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa']
keys = [ species.name_suggest(x)['data'][0]['key'] for x in splist ]
occ.search(taxonKey = keys, limit = 5)
```

249 Ruby

```
species = Gbif::Species
occ = Gbif::Occurrences

splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa']
keys = [ species.name_suggest(x)['data'][0]['key'] for x in splist ]
occ.search(taxonKey: keys, limit: 5)
```

250 Spatial search, based on well known text format (Herring, 2011), or a bounding box set of four co-
 251 ordinates. The well known text string and the bounding box in the below example specify the same
 252 rectangular area in California, centering approximately on Sacramento. Whereas the bounding box for-
 253 mat requires longitude SW corner, latitude SW corner, longitude NE corner, latitude NE
 254 corner, the well known text string requires an extra long/lat pair to close the polygon.

255 **R**

```
# well known text
wkt <- 'POLYGON((-122.6 39.9,-120.0 39.9,-120.0 37.9,-122.6 37.9,-122.6 39.9))'
occ_search(geometry = wkt, limit = 20)
# bounding box
occ_search(geometry = c(-122.6,37.9,-120.0,39.9), limit = 20)
```

256 **Python**

```
from pygbif import occurrences as occ
# well known text
occ.search(geometry = 'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit = 20)
# bounding box
occ.search(geometry = '-125.0,38.4,-121.8,40.9', limit = 20)
```

257 **Ruby**

```
occ = Gbif::Occurrences
# well known text
occ.search(geometry: 'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit: 20)
# bounding box
occ.search(geometry: '-125.0,38.4,-121.8,40.9', limit: 20)
```

258 Get only occurrences with lat/long data using the `hasCoordinate` parameter

259 **R**

```
occ_search(hasCoordinate = TRUE, limit = 5)
```

260 **Python**

```
from pygbif import occurrences as occ
occ.search(hasCoordinate = True, limit = 5)
```

261 **Ruby**

```
occ = Gbif::Occurrences
occ.search(hasCoordinate: true, limit: 5)
```

262 Get only those occurrences with spatial issues. Spatial issues are a set of issues that are returned in
 263 the `issues` field. They each indicate something different about that record. For example, the issue
 264 `COUNTRY_COORDINATE_MISMATCH` indicates that the interpreted occurrence coordinates fall outside of
 265 the indicated country. You can see how that might be useful when it comes to cleaning your data prior
 266 to analysis/visualization.

267 **R**

```
occ_search(hasGeospatialIssue = TRUE, limit = 5)
```

268 **Python**

```
from pygbif import occurrences as occ
occ.search(hasGeospatialIssue = True, limit = 5)
```

269 **Ruby**

```
occ = Gbif::Occurrences
occ.search(hasGeospatialIssue: true, limit: 5)
```

270 *Data cleaning.* GBIF provides optional data issues with each occurrence record. These issues fall into
 271 many different pre-defined classes, covering issues with taxonomic names, geographic data, and more
 272 (see `rgbif::occ_issues_lookup()` to find out more information on GBIF issues; and the same data
 273 on [GBIF's development site](#)).

274 `rgbif::occ_issues()` provides a way to easily filter data downloaded via `rgbif::occ_search()` based
 275 on GBIF issues.

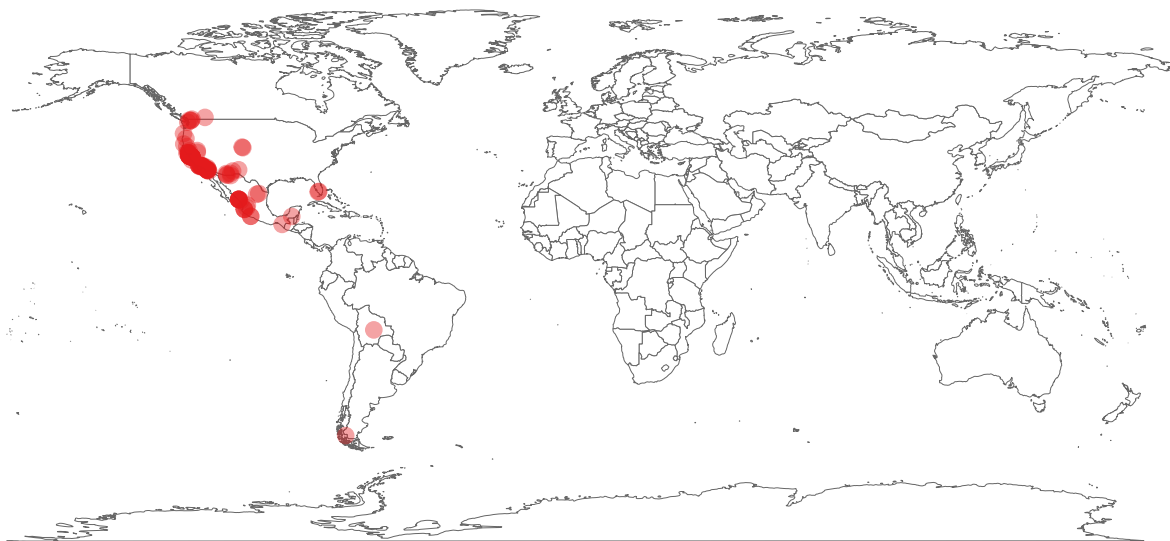
```
out <- occ_search(issue = 'DEPTH_UNLIKELY', limit = 500)
NROW(out)
#> [1] 5
out %>% occ_issues(-cudc) %>% .$data %>% NROW
#> [1] 0
```

276 There's no equivalent interface in `pygbif` or `gbifrb` yet.

277 *Mapping*

278 An obvious downstream use case for species occurrence data is to map the data. `rgbif` per se is largely
279 not concerned with making this easier, although we do have a simple wrapper around `ggplot2` to make
280 it easy to get a quick plot of occurrence data. For example, here we plot 100 occurrences for *Puma*
281 *concolor*.

```
key <- name_backbone(name='Puma concolor')$speciesKey
dat <- occ_search(taxonKey = key, limit = 100, hasCoordinate = TRUE)
gbifmap(dat$data)
```



282

283 Another package, `mapr`, is the perfect mapping companion to `rgbif`. It has convenient functions for
284 handling input data from `rgbif`, `spocc`, or arbitrary `data.frame`'s, and output plots for base plots,
285 `ggplot2`, `ggmap` (`ggplot2` with map layers underneath), and interactive maps on GitHub gists or with
286 Leaflet.js.

287 There's no equivalent interface in `pygbif` or `gbifrb`.

288 *GBIF data in other R packages*

289 We discuss usage of GBIF data in other R packages throughout the manuscript, but provide a synopsis
290 here for clarity.

291 *taxize*

292 Some of the GBIF taxonomic services are also available in [taxize](#), an R package that focuses on getting
293 data from taxonomic data sources on the web. For example, with `get_gbifid()` one can get GBIF IDs
294 used for a set of taxonomic names - then use those IDs in other functions in `taxize` to get additional
295 information, like taxonomically downstream children.

296 *spocc*

297 GBIF occurrence data is available in the R package [spocc](#) via `rgbif`. `spocc` is a unified interface
298 for fetching species occurrence data from many sources on the web. For example, a user can collect
299 occurrence data from GBIF, iDigBio, and iNaturalist, and easily combine them, then use other packages
300 to clean and visualize the data.

301 **R vs. Python vs. Ruby**

302 Both R and Python are commonly used in science, and can be used for similar tasks. Python, however,
303 is a more general programming language, and can be used in more contexts than R can be used in.
304 Ruby is used very little in science; but, like Python, Ruby is very widely used as a general purpose
305 programming language, with heavy use in web development and web services.

306 The three clients can do a lot of the same tasks. We envision `rgbif` being more common in workflows
307 of academics asking research questions, whereas `pygbif` and `gbifrb` can do that as well, but may be
308 more easily used in a website.

309 The R client `rgbif` has had much more development time than `pygbif` and `gbifrb`, but with time
310 `pygbif` and `gbifrb` will become equally mature.

311 **Use cases**

312 The following are three use cases for the R library `rgbif`: niche modeling, spatial change in biodiversity,
313 and distribution mapping.

314 *Ecological niche modeling*

315 In this example, we plot actual occurrence data for *Bradypus* species against a single predictor variable,
316 BIO1 (annual mean temperature). This is only one step in a species distribution modelling workflow.
317 This example can be done using BISON data as well with our rbison package.

318 *Load libraries*

```
library("sp")
library("rgbif")
library("dismo")
library("maptools")
library("dplyr")
```

319 *Raster files*

320 Make a list of files that are installed with the dismo package, then create a rasterStack from these

```
files <- list.files(paste(system.file(package = "dismo"), "/ex", sep = ""),
                   "grd", full.names = TRUE)
predictors <- stack(files)
```

321 *Get world boundaries*

```
data(wrld_simpl)
```

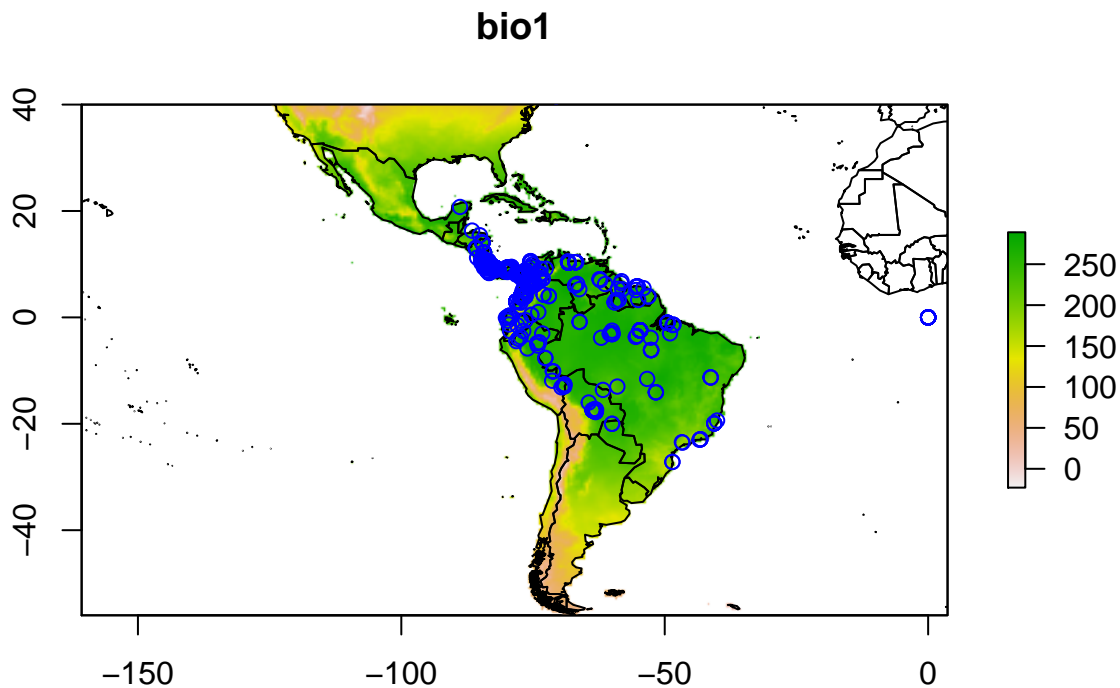
322 *Get GBIF data using the rOpenSci package rgbif*

```
nn <- name_lookup("bradypus*", rank = "species")
nn <- na.omit(unique(nn$data$subKey))
df <- occ_search(taxonKey = nn, hasCoordinate = TRUE, limit = 500)
df_data <- df[ apply(df, function(x) any(class(x$data) %in% "tbl_df")) ]
df_data <- dplyr::bind_rows(lapply(df_data, "[", "data"))
df2 <- df_data %>% dplyr::select(decimalLongitude, decimalLatitude)
```

323 *Plot*

324 (1) Add raster data, (2) Add political boundaries, (3) Add the points (occurrences)

```
plot(predictors, 1)
plot(wrld_simpl, add = TRUE)
points(df2, col = "blue")
```



325

326 *Biodiversity in big cities*

327 In this example, we collect specimen records across different cities using GBIF data from the `rgbif`
328 package.

329 *Load libraries*

```
library("rgbif")
library("ggplot2")
library("plyr")
library("httr")
```

```
library("RColorBrewer")
library("wicket")
```

330 *Get bounding boxes for some cites*

331 Bounding lat/long data is from [https://raw.githubusercontent.com/amyxzhang/boundingbox-cities/master/](https://raw.githubusercontent.com/amyxzhang/boundingbox-cities/master/boundbox.txt)
 332 [boundbox.txt](https://raw.githubusercontent.com/amyxzhang/boundingbox-cities/master/boundbox.txt).

```
url <- 'https://raw.githubusercontent.com/amyxzhang/
boundingbox-cities/master/boundbox.txt'
rawdat <- content(GET(sub("\n", "", url)), as = "text")
dat <- read.table(
  text = rawdat, header = FALSE,
  sep = "\t", col.names = c("city", "minlat", "maxlon", "maxlat", "minlon"),
  stringsAsFactors = FALSE)
dat <- data.frame(
  city = dat$city, minlon = dat$minlon,
  minlat = dat$minlat, maxlon = dat$maxlon,
  maxlat = dat$maxlat,
  stringsAsFactors = FALSE
)
```

333 A helper function to get count data. GBIF has a count API, but we can't use that with a geometry search
 334 as that API doesn't support geospatial search. We can however use the search API via `occ_search()`
 335 and set `limit = 1` so that we

```
getdata <- function(x){
  coords <- as.numeric(x[c('minlon', 'minlat', 'maxlon', 'maxlat')])
  wkt <- wicket::wkt_correct(wicket::bounding_wkt(values = coords))
  num <- occ_search(geometry = wkt, limit = 1)$meta$count
  data.frame(
    city = x['city'],
    richness = num,
```



```

    stringsAsFactors = FALSE
  )
}

```

```

out <- apply(dat, 1, getdata)

```

336 *Merge to original table*

```

out <- merge(dat, ldply(out), by = "city")

```

337 *Add centroids from bounding boxes*

```

out <- transform(out, lat = (minlat + maxlat)/2, lon = (minlon + maxlon)/2)

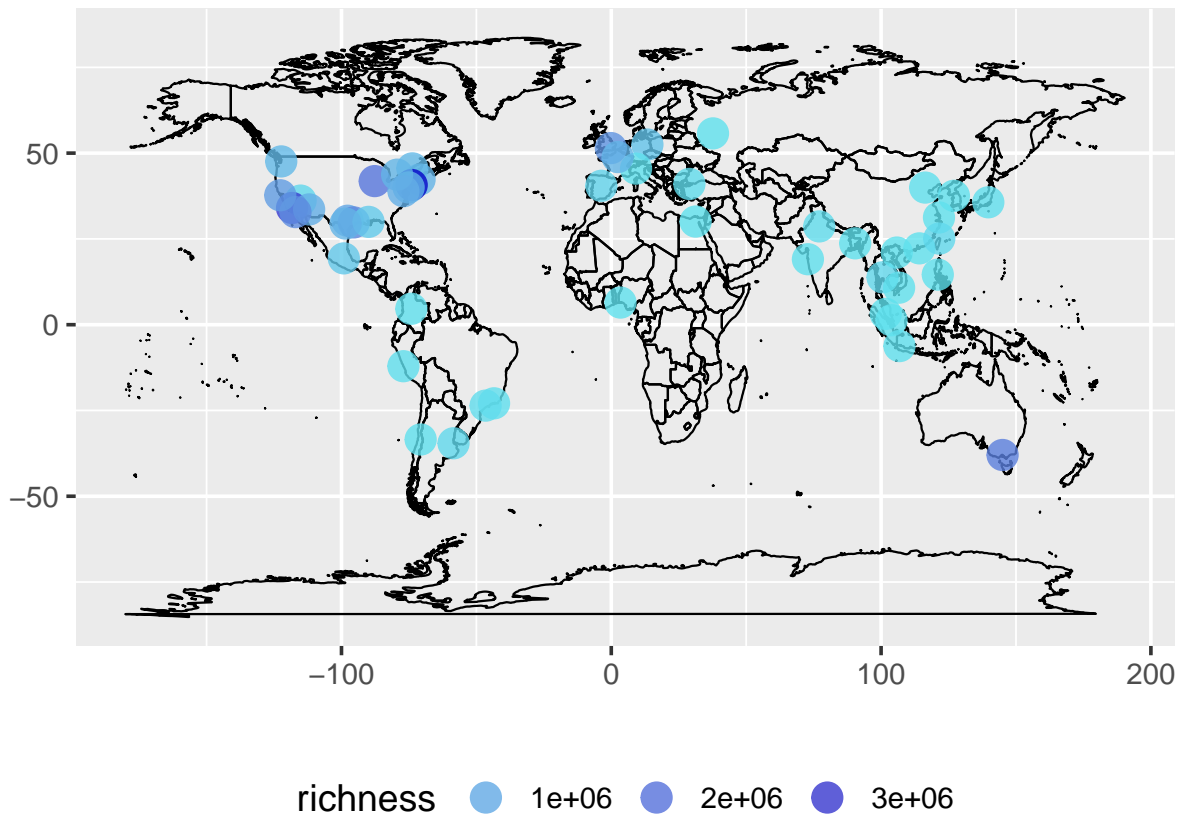
```

338 *Plot data*

```

mapp <- map_data('world')
ggplot(mapp, aes(long, lat)) +
  geom_polygon(aes(group=group), fill="white", alpha=0, color="black", size=0.4) +
  geom_point(data=out, aes(lon, lat, color=richness), size=5, alpha=0.8) +
  scale_color_continuous(low = "#60E1EE", high = "#0404C8") +
  labs(x="", y="") +
  theme_grey(base_size=14) +
  theme(legend.position = "bottom", legend.key = element_blank()) +
  guides(color = guide_legend(keywidth = 2))

```



339

340 *Valley oak occurrence data comparison*

341 This example is inspired by a tweet from [Antonio J. Perez-Luque](#) who [shared his plot on Twitter](#).
 342 Antonio compared the occurrences of Valley Oak (*Quercus lobata*) from [GBIF](#) to the distribution of the
 343 same species from the [Atlas of US Trees](#).

344 The data in question from the example above is no longer available, so below we use a different species.

345 *Load libraries*

```
library('rgbif')
library('raster')
library('sp')
library('sf')
library('rgeos')
library('scales')
library('rnaturalearth')
```

346 *Get GBIF Data for Fraxinus excelsior*

```
keyFe <- name_backbone(name = 'Fraxinus excelsior', kingdom = 'plants')$speciesKey
dat.Fe <- occ_search(taxonKey = keyFe, return = 'data', limit = 10000L)
```

347 *Get Distribution map of F. excelsior European Forest Genetic Resources Programme*

348 From <http://www.euforgen.org/species/fraxinus-excelsior/>. And save shapefile in same directory

```
url <- 'http://www.euforgen.org/fileadmin/templates/euforgen.org/upload/Documents/Maps/Shapefile'
tmp <- tempdir()
download.file(url, destfile = "fraxinus_excelsior.zip")
unzip("fraxinus_excelsior.zip", exdir = tmp)
fe <- sf::read_sf(file.path(tmp, "Fraxinus_excelsior_EUFORGEN.shp"))
```

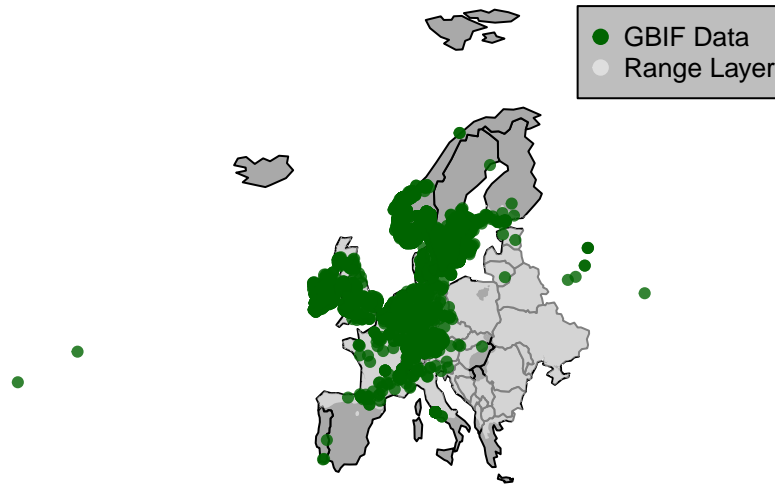
349 *Get Elevation data of US*

```
eur <- rnaturalearth::ne_countries(continent = "europe", type = "map_units")
eur1 <- eur[eur$sovereignty != "Russia", ]
```

350 *Plot map*

```
plot(eur1, col = "darkgrey", legend = FALSE,
     main = 'Distribution of Fraxinus excelsior')
# add distribution range layer
plot(fe, add = TRUE, col = alpha("white", 0.5), border = FALSE)
# add Gbif presence points
points(dat.Fe$decimalLongitude, dat.Fe$decimalLatitude,
       cex = .7, pch = 19, col = alpha("darkgreen", 0.8))
legend(x = 38, y = 81, c("GBIF Data", "Range Layer"), pch = 19, bg = "grey",
      col = c('darkgreen', alpha("white", 0.5)), pt.cex = 1, cex = .8)
```

Distribution of *Fraxinus excelsior*



351

352 Conclusions and future directions

353 The `rgbif`, `pygbif`, and `gbifrb` libraries provide programmatic interfaces to GBIF's application
354 programming interface (API) - a powerful tool for working with species occurrence data, and facilitating
355 reproducible research. In fact, the `rgbif` package has already been used in more than 20 scholarly
356 publications (as of 2016-08-10), including (Amano, Lamming & Sutherland, 2016, Bartomeus et al.
357 (2013), Barve (2014), Bone et al. (2015), Collins et al. (2015), Drozd & Šipoš (2013), Kong, Huang &
358 Duan (2015), Richardson, Roux & Wilson (2015), Turner, Fréville & Rieseberg (2015), Verheijen et al.
359 (2015), Zizka & Antonelli (2015), Butterfield et al. (2016), Dellinger et al. (2015), Feitosa et al. (2015),
360 Malhado et al. (2015), Werner et al. (2015), Robertson, Visser & Hui (2016), Davison et al. (2015),
361 Janssens et al. (2016)).

362 The `rgbif` package is relatively stable, and should not have many breaking changes unless necessitated
363 due to changes in the GBIF API. However, it will gain function(s) to work with the maps API in the
364 near future.

365 The `pygbif` and `gbifrb` libraries are in early development, and will greatly benefit from any feedback
366 and use cases.

367 One area of focus in the future is to attempt to solve many use cases that have been brought up with
368 respect to GBIF data. For example, some specimens are included in GBIF that are located in botanical
369 gardens. For many research questions, researchers are interested in “wild” type occurrences, not those
370 in human curated scenarios. Making removal of these occurrences easy would be very useful, but is
371 actually quite a hard problem. There are many other problems like this, for which these three libraries
372 will help in making more efficient and reproducible.

373 Acknowledgments

374 This project was supported in part by the Alfred P Sloan Foundation (Grant No. G-2014-13485), and
375 in part by the Helmsley Foundation (Grant No. 2016PG-BRI004).

376 Data Accessibility

377 All scripts and data used in this paper can be found in the permanent data archive Zenodo under
378 the digital object identifier (<https://doi.org/10.5281/zenodo.997547>). This DOI corresponds to a
379 snapshot of the GitHub repository at <https://github.com/sckott/gbifms> that matches this preprint.
380 Software can be found at <https://github.com/ropensci/rgbif>, <https://github.com/sckott/pygbif>, and
381 <https://github.com/sckott/gbifrb>, all under MIT licenses. We thank all the users that have used `rgbif`,
382 `pygbif`, and `gbifrb` and have given feedback and reported bugs. In addition, we greatly appreciate all
383 the contributors to the three libraries, found at <https://github.com/ropensci/rgbif/graphs/contributors>,
384 <https://github.com/sckott/pygbif/graphs/contributors>, and [https://github.com/sckott/gbifrb/graphs/](https://github.com/sckott/gbifrb/graphs/contributors)
385 [contributors](https://github.com/sckott/gbifrb/graphs/contributors).

386 References

387 Amano T., Lamming JDL., Sutherland WJ. 2016. Spatial gaps in global biodiversity information and
388 the role of citizen science. *BioScience* 66:393–400.

389 Bartomeus I., Park MG., Gibbs J., Danforth BN., Lakso AN., Winfree R. 2013. Biodiversity ensures

390 plant-pollinator phenological synchrony against climate change. *Ecology Letters* 16:1331–1338.

391 Barve V. 2014. Discovering and developing primary biodiversity data from social networking sites: A
392 novel approach. *Ecological Informatics* 24:194–199.

393 Beck J., Ballesteros-Mejia L., Buchmann CM., Dengler J., Fritz SA., Gruber B., Hof C., Jansen
394 F., Knapp S., Kreft H., Schneider A-K., Winter M., Dormann CF. 2012. Whats on the horizon for
395 macroecology? *Ecography* 35:673–683.

396 Bone RE., Smith JAC., Arrigo N., Buerki S. 2015. A macro-ecological perspective on crassulacean acid
397 metabolism (CAM) photosynthesis evolution in afro-madagascan drylands: Eulophiinae orchids as a
398 case study. *New Phytologist* 208:469–481.

399 Brown JH. 1995. *Macroecology*. University of Chicago Press.

400 Brown KA., Parks KE., Bethell CA., Johnson SE., Mulligan M. 2015. Predicting plant diversity patterns
401 in madagascar: Understanding the effects of climate and land cover change in a biodiversity hotspot.
402 *PLOS ONE* 10:e0122721.

403 Butterfield BJ., Copeland SM., Munson SM., Roybal CM., Wood TE. 2016. Prestoration: Using species
404 in restoration that will persist now and into the future. *Restor Ecol.*

405 Ceballos G., Ehrlich PR., Barnosky AD., Garcia A., Pringle RM., Palmer TM. 2015. Accelerated
406 modern human-induced species losses: Entering the sixth mass extinction. *Science Advances* 1:e1400253–
407 e1400253.

408 Chamberlain S., Ram K., Barve V., Mcglinn D. *rgbif: An r interface to the global 'biodiversity'
409 information facility API*.

410 Chamberlain S. *pygbif: A python interface to the global biodiversity information facility API*.

411 Chamberlain S. *gbifrb: A ruby interface to the global biodiversity information facility API*.

412 Collins R., Ribeiro ED., Machado VN., Hrbek T., Farias I. 2015. A preliminary inventory of the catfishes
413 of the lower rio nhamundá, brazil (ostariophysi, siluriformes). *BDJ* 3:e4162.

414 Davison J., Moora M., Opik M., Adholeya A., Ainsaar L., Ba A., Burla S., Diedhiou AG., Hiiesalu
415 I., Jairus T., Johnson NC., Kane A., Koorem K., Kochar M., Ndiaye C., Partel M., Reier U., Saks
416 U., Singh R., Vasar M., Zobel M. 2015. Global assessment of arbuscular mycorrhizal fungus diversity
417 reveals very low endemism. *Science* 349:970–973.

418 Dellinger AS., Essl F., Hojsgaard D., Kirchheimer B., Klatt S., Dawson W., Pergl J., Pyšek P., Kleunen

419 M van., Weber E., Winter M., Hörandl E., Dullinger S. 2015. Niche dynamics of alien species do not
420 differ among sexual and apomictic flowering plants. *New Phytologist* 209:1313–1323.

421 Drozd P., Šipoš J. 2013. R for all (i): Introduction to the new age of biological analyses. *Casopis
422 slezského zemského muzea (A)* 62.

423 Faulkner KT., Robertson MP., Rouget M., Wilson JR. 2014. A simple, rapid methodology for developing
424 invasive species watch lists. *Biological Conservation* 179:25–32.

425 Febbraro MD., Lurz PWW., Genovesi P., Maiorano L., Girardello M., Bertolino S. 2013. The use of
426 climatic niches in screening procedures for introduced species to evaluate risk of spread: A case with
427 the american eastern grey squirrel. *PLoS ONE* 8:e66559.

428 Feitosa YO., Absy ML., Latrubesse EM., Stevaux JC. 2015. Late quaternary vegetation dynamics from
429 central parts of the madeira river in brazil. *Acta Bot. Bras.* 29:120–128.

430 Ferretti F., Verd GM., Seret B., Šprem JS., Micheli F. 2015. Falling through the cracks: The fading
431 history of a large iconic predator. *Fish and Fisheries:n/a–n/a*.

432 Ficetola GF., Rondinini C., Bonardi A., Baisero D., Padoa-Schioppa E. 2014. Habitat availability for
433 amphibians and extinction threat: A global analysis. *Diversity and Distributions* 21:302–311.

434 Herring J. 2011. OpenGIS implementation standard for geographic information-simple feature access-
435 part 1: Common architecture. *OGC Document* 4:122–127.

436 Janssens SB., Vandeloek F., Langhe ED., Verstraete B., Smets E., Vandenhoutte I., Swennen R. 2016.
437 Evolutionary dynamics and biogeography of musaceae reveal a correlation between the diversification
438 of the banana family and the geological and climatic history of southeast asia. *New Phytologist*
439 210:1453–1465.

440 Kong X., Huang M., Duan R. 2015. SDMdata: A web-based software tool for collecting species
441 occurrence records. *PLOS ONE* 10:e0128295.

442 Malhado AC., Oliveira-Neto JA., Stropp J., Strona G., Dias LC., Pinto LB., Ladle RJ. 2015. Climato-
443 logical correlates of seed size in amazonian forest trees. *J Veg Sci* 26:956–963.

444 María Mendoza., Ospina OE., Cárdenas-Henao H., García-R JC. 2015. A likelihood inference of
445 historical biogeography in the world’s most diverse terrestrial vertebrate genus: Diversification of
446 direct-developing frogs (craugastoridae: Pristimantis) across the neotropics. *Molecular Phylogenetics*

447 *and Evolution* 85:50–58.

448 Pimm SL., Jenkins CN., Abell R., Brooks TM., Gittleman JL., Joppa LN., Raven PH., Roberts CM.,
449 Sexton JO. 2014. The biodiversity of species and their rates of extinction, distribution, and protection.
450 *Science* 344:1246752–1246752.

451 R Core Team. 2014. *R: A language and environment for statistical computing*. Vienna, Austria: R
452 Foundation for Statistical Computing.

453 Richardson DM., Roux J.J.L., Wilson J.R. 2015. Australian acacias as invasive species: Lessons to be
454 learnt from regions with long planting histories. *Southern Forests: a Journal of Forest Science* 77:31–39.

455 Robertson MP., Visser V., Hui C. 2016. Biogeo: An r package for assessing and improving data quality
456 of occurrence record datasets. *Ecography* 39:394–401.

457 Turner K.G., Fréville H., Rieseberg L.H. 2015. Adaptive plasticity and niche expansion in an invasive
458 thistle. *Ecol Evol* 5:3183–3197.

459 Verheijen L.M., Aerts R., Bönisch G., Kattge J., Bodegom P.M.V. 2015. Variation in trait trade-offs
460 allows differentiation among predefined plant functional types: Implications for predictive ecology. *New*
461 *Phytologist* 209:563–575.

462 Werner G.D.A., Cornwell W.K., Cornelissen J.H.C., Kiers E.T. 2015. Evolutionary signals of symbiotic
463 persistence in the legume-rhizobia mutualism. *Proceedings of the National Academy of Sciences* 112:10262–
464 10269.

465 Zizka A., Antonelli A. 2015. *speciesgeocodeR: An r package for linking species occurrences, user-defined*
466 *regions and phylogenetic trees for biogeography, ecology and evolution*. Cold Spring Harbor Laboratory
467 Press.