

R and Python 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.

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

17 Results. For both clients, we describe their design and utility, and demonstrate many use cases.

Discussion. Programmatic access to GBIF will facilitate more open and reproducible science. We hope
that all scientists using GBIF data will do so with one of these two clients.

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 643 million records, 1.6 million taxa, 15,450 datasets from 780 publishers
38 (as of 2016-02-09). Many large biodiversity warehouses such as iNaturalist (<http://www.inaturalist.org>),
39 VertNet (<http://vertnet.org>), and USGS's Biodiversity Information Serving Our Nation (BISON;
40 <http://bison.usgs.ornl.gov>) all feed into GBIF.

41 The most important data organizational level in GBIF is the individual record. In R using `rgbif`
42 you'll recognize this as a row in the data.frame that is returned from `occ_search()` or `occ_data()`.
43 Going upstream, each record is part of a dataset, where each dataset is submitted by an organization,
44 organizations are organized into nodes, datasets are published through institutions (which may be
45 hosted at another organization), and a network is a group of datasets (managed by GBIF).

Each record has some taxonomic name associated with it, which itself is linked to a lot of other taxonomic data. GBIF maintains their own taxonomic data.

This organization matters because you can navigate through GBIF data through these various organizational levels.

The clients

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
```

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

Library interfaces

`rgbif` and `pygbif` are designed following the [GBIF Application Programming Interface](#), or API. The GBIF API has four major components: registry, taxonomic names, occurrences, and maps. We also include functions to interface with the OAI-PMH GBIF service; only dataset information is available vis this service, however. We ignore maps in both libraries as it is concerned with generating maps primarily for web applications. Both libraries have a suite of functions dealing with each of registry, taxonomic names, and occurrences - we'll go through each in turn describing design and example usage.

GBIF headers

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

Registry

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

- Datasets
- Installations
- Networks
- Nodes
- Organizations

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

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

100 *Datasets*

101 Dataset functions include search, dataset metadata retrieval, and dataset metrics. Searching for datasets
102 is an important part of the discovery process. One can search for datasets on the GBIF web portal.
103 However, programmatic searching using this package is much more powerful. Identifying datasets
104 appropriate for a research question is helpful as you can get metadata for each dataset, and track down
105 dataset specific problems, if any.

106 The `dataset_search()` function is one way to search for datasets. Here, we search for the query term
107 “oregon”, which finds any datasets that have terms matching that term.

```
res <- dataset_search(query = "oregon")
res$data$datasetTitle[1:10]
#> [1] "UCDavis - Western USA - Monarch Butterflies - 1892-2005"
#> [2] "A geographic distribution database of Mononychellus mites (Acari: Tetranychidae) on cas
#> [3] "Plantas Acuáticas de la Orinoquía Colombiana"
#> [4] "USBombus, contemporary survey data of North American bumble bees (Hymenoptera, Apidae,
#> [5] "Wool carder bees of the genus Anthidium in the Western Hemisphere"
#> [6] "CM Birds Collection"
#> [7] "SDNHM Birds Collection"
#> [8] "University of British Columbia Herbarium (UBC) - Bryophytes Collection"
#> [9] "University of British Columbia Herbarium (UBC) - Vascular Plant Collection"
#> [10] "Bryophyte Collection - University of Washington Herbarium (WTU)"
```

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

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

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

110 *Dataset metrics.* Dataset metrics are another useful way of figuring out what datasets you may want to
 111 use. One drawback is that these metrics data are only available for datasets of type *checklist*, but there
 112 are quite a lot of them (16084).

113 Here, we search for dataset metrics for a single dataset, with uuid `ec93a739-1681-4b04-b62f-3a687127a17f`,
 114 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

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

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

116 *Networks, nodes, and installations*

117 Networks, nodes and installations are at a higher level of organization above datasets, but can be
118 useful if you want to explore data from given organizations. Here, in R we search for the first 10 GBIF
119 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)"
```

120 And in Python:

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

121 *Taxonomic names*

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

- 123 • Search GBIF name backbone - `name_backbone()`
- 124 • Search across all checklists - `name_lookup()`
- 125 • Quick name lookup - `name_suggest()`
- 126 • Name usage of a name according to a checklist - `name_usage()`
- 127 • GBIF name parser - `parsenames()`

128 `pygbif` only has `name_backbone()` and `name_suggest()` at this time.

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

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

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

140 *GBIF Backbone*

141 The GBIF backbone taxonomy is used in GBIF to have a consistent way to refer to taxonomic
142 names throughout their services. The backbone has 5296599 unique names and 2525274 species
143 names. The backbone taxonomy is also a dataset with key `d7dddbf4-2cf0-4f39-9b2a-bb099caae36c`
144 (<http://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c>).

145 We can search the backbone taxonomy with the function `name_backbone()` in both R and Python
146 clients. Here, 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"
```

147 and in Python

```

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

```

148 *Name searching*

149 One of the quickest ways to search for names is using `name_suggest()`, which does a very quick search
 150 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
4848380		Pumiliornis	GENUS
2235094		Pumilibranchipus	GENUS
1795637		Pumora	GENUS
4581745		Pumicia	GENUS
4598790		Pumilicopta	GENUS
1593095		Pumilomyia	GENUS

151 The same in Python

```

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

```

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

154 *Occurrences*

155 GBIF provides two ways to get occurrence data: through the `/occurrence/search` route (see
 156 `occ_search` in `rgbif`, or `occurrences.search` in `pygbif`), or via the `/occurrence/download` route
 157 (many functions, see below). `occ_search()/occurrences.search` is the main function for the search
 158 route, and is more appropriate for smaller data, while the download functions are more appropriate for
 159 larger data requests.

160 GBIF imposes for any given search a limit of 200,000 records in the search service, after which point
161 you can't download any more records for that search. However, you can download more records for
162 different searches.

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

167 The download service, like the occurrence search service, is rate-limited. That is, you can only have one
168 to three downloads running simultaneously for your user credentials. However, simply check when a
169 download job is complete, then you should be able to start a new download request.

170 *Download API*

171 The download API syntax is similar to the occurrence search API in that the same parameters are
172 used, but the way in which the query is defined is different. For example, in the download API you can
173 do greater than searches (i.e., `latitude > 50`), whereas you cannot do that in the occurrence search
174 API. Thus, unfortunately, we couldn't make the query interface exactly the same for both search and
175 download functions.

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

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

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

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

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

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

185 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
```

186 In Python

```

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

```

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

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

194 *Search API*

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

- 196 • Get a single numeric count of occurrences - `rgbif: occ_count()` / `pygbif: occurrences.count`
- 197 • Search for occurrences - `rgbif: occ_search()` / `pygbif: occurrences.count`
- 198 • A simplified and optimized version of `rgbif: occ_search()` or `occ_data()` / `pygbif:`
 199 `occurrences.count`
- 200 • Get occurrences by occurrence identifier - `rgbif: occ_get()` / `pygbif: occurrences.count`
- 201 • Get occurrence metadata - `rgbif: occ_metadata()` / `pygbif: occurrences.count`

202 *Search for occurrences. R*

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

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

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

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

```
(key <- name_suggest(q = 'Helianthus annuus', rank = 'species')$key[1])
#> [1] 3119195
occ_search(taxonKey = key, limit = 2)
#> Records found [21538]
#> Records returned [2]
#> No. unique hierarchies [1]
#> No. media records [2]
#> Args [taxonKey=3119195, limit=2, offset=0, fields=all]
#> First 10 rows of data
#>
#>           name           key decimalLatitude decimalLongitude
#> 1 Helianthus annuus 1249279611          34.04810         -117.79884
#> 2 Helianthus annuus 1249286909          32.58747          -97.10081
#> Variables not shown: issues (chr), datasetKey (chr), publishingOrgKey
#> (chr), publishingCountry (chr), protocol (chr), lastCrawled (chr),
#> lastParsed (chr), extensions (chr), basisOfRecord (chr), taxonKey
#> (int), kingdomKey (int), phylumKey (int), classKey (int), orderKey
#> (int), familyKey (int), genusKey (int), speciesKey (int),
#> scientificName (chr), kingdom (chr), phylum (chr), order (chr),
#> family (chr), genus (chr), species (chr), genericName (chr),
#> specificEpithet (chr), taxonRank (chr), dateIdentified (chr), year
#> (int), month (int), day (int), eventDate (chr), modified (chr),
#> lastInterpreted (chr), references (chr), identifiers (chr), facts
#> (chr), relations (chr), geodeticDatum (chr), class (chr), countryCode
#> (chr), country (chr), rightsHolder (chr), identifier (chr),
#> verbatimEventDate (chr), datasetName (chr), gbifID (chr),
#> verbatimLocality (chr), collectionCode (chr), occurrenceID (chr),
#> taxonID (chr), license (chr), recordedBy (chr), catalogNumber (chr),
#> http...unknown.org.occurrenceDetails (chr), institutionCode (chr),
```

```
#>      rights (chr), eventTime (chr), identificationID (chr),
#>      coordinateAccuracy (dbl), coordinateAccuracyInMeters (dbl),
#>      occurrenceRemarks (chr)
```

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

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

214 Search for many species

```
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')
```

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

```
# 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)
```

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

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

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

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

226 Python

227 The equivalent occurrence search workhorse in pygibf is `occurrences.search`.

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

```
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)
```

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

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

230 Search for many species

```
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)
```

231 Spatial search, based on well known text format (Herring, 2011), or a bounding box set of four coordinates

```
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)
```

232 Get only occurrences with lat/long data


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

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

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

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

242 `occ_issues()` provides a way to easily filter data downloaded via `occ_search()` based on GBIF issues.

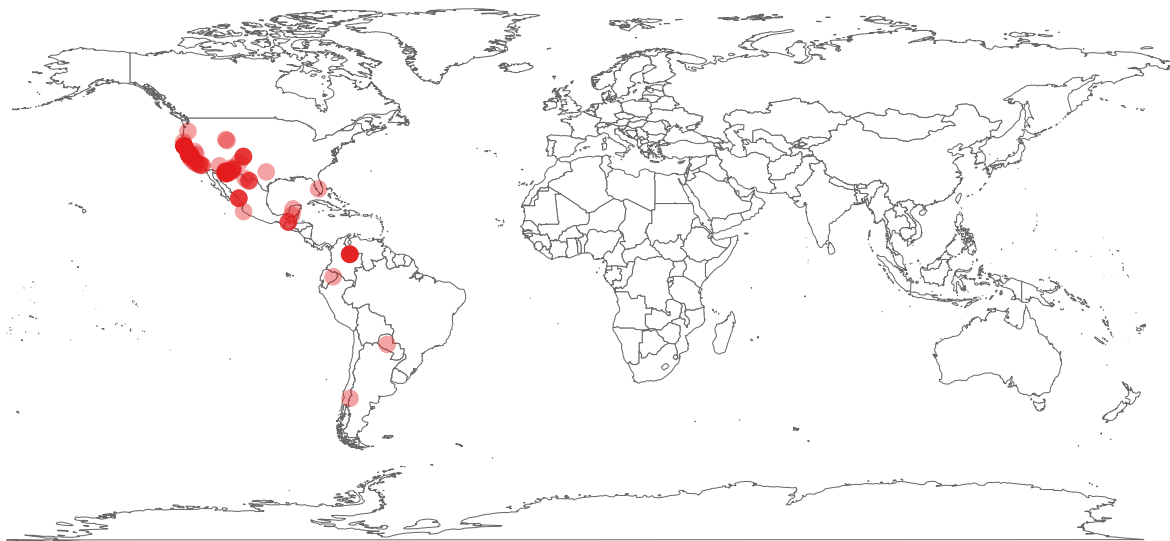
```
out <- occ_search(issue = 'DEPTH_UNLIKELY', limit = 500)
NROW(out)
#> [1] 4
out %>% occ_issues(-cudc) %>% .$data %>% NROW
#> [1] 4
```

243 There's no equivalent interface in `pygbif` yet.

244 *Mapping*

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

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



249

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

254 There's no equivalent interface in `pygbif`.

255 *GBIF data in other R packages*

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

258 *taxize*

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

263 *spocc*

264 GBIF occurrence data is available in the R package `spocc` via `rgbif`. `spocc` is a unified interface
265 for fetching species occurrence data from many sources on the web. For example, a user can collect
266 occurrence data from GBIF, iDigBio, and iNaturalist, and easily combine them, then use other packages
267 to clean and visualize the data.

268 **R vs. Python**

269 Both R and Python are commonly used in science, and can be used for similar tasks. Python, however,
270 is a more general programming language, and can be used in more contexts than R can be used in.

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

274 The R client `rgbif` has much more development time than `pygbif`, but through time `pygbif` will
275 become equally mature.

276 **Use cases**

277 The following are three use cases for `rgbif`: niche modeling, spatial change in biodiversity, and
278 distribution mapping.

279 *Ecological niche modeling*

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

283 *Load libraries*

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

284 *Raster files*

285 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)
```

286 *Get world boundaries*

```
data(wrld_simpl)
```

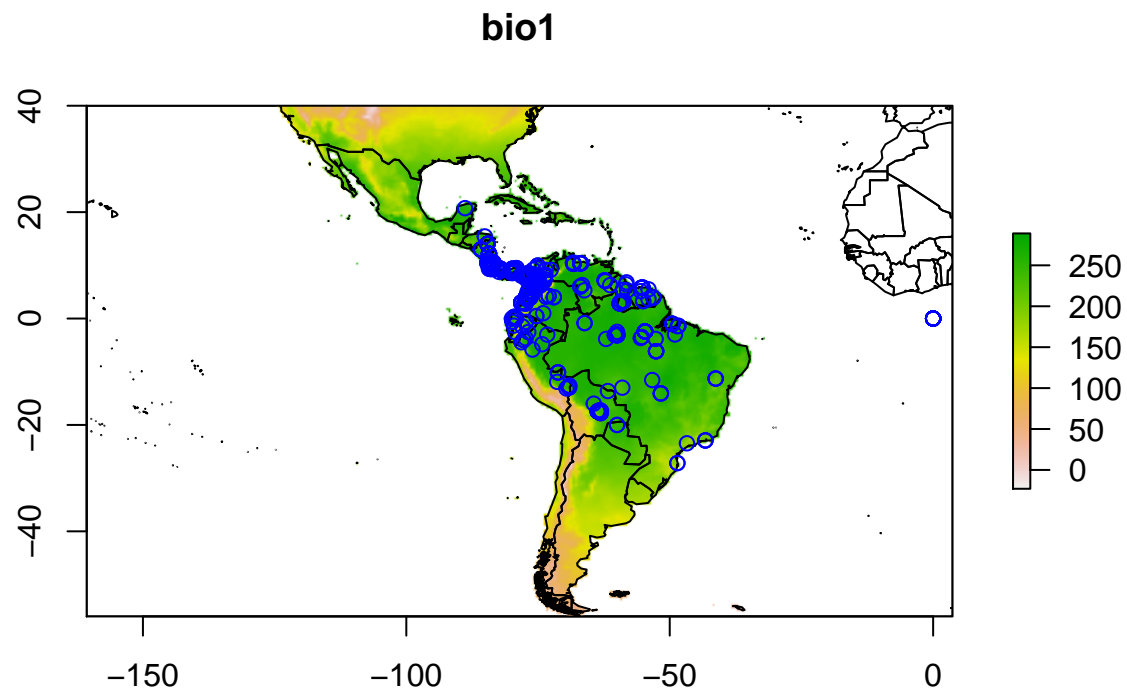
287 *Get GBIF data using the `rOpenSci` package `rgbif`*

```
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[ sapply(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)
```

288 *Plot*

289 (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")
```



290

291 *Biodiversity in big cities*

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

294 *Load libraries*

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

295 *Get bounding boxes for some cities*

296 Bounding lat/long data is from [https://raw.githubusercontent.com/amyxzhang/boundingbox-cities/master/](https://raw.githubusercontent.com/amyxzhang/boundingbox-cities/master/boundbox.txt)
297 [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
)

```

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

```

getdata <- function(x){
  coords <- as.numeric(x[c('minlon', 'minlat', 'maxlon', 'maxlat')])
  num <- occ_search(geometry = coords, limit = 1)$meta$count
  data.frame(
    city = x['city'],
    richness = num,
    stringsAsFactors = FALSE
  )
}

```

```

out <- apply(dat, 1, getdata)

```

301 *Merge to original table*

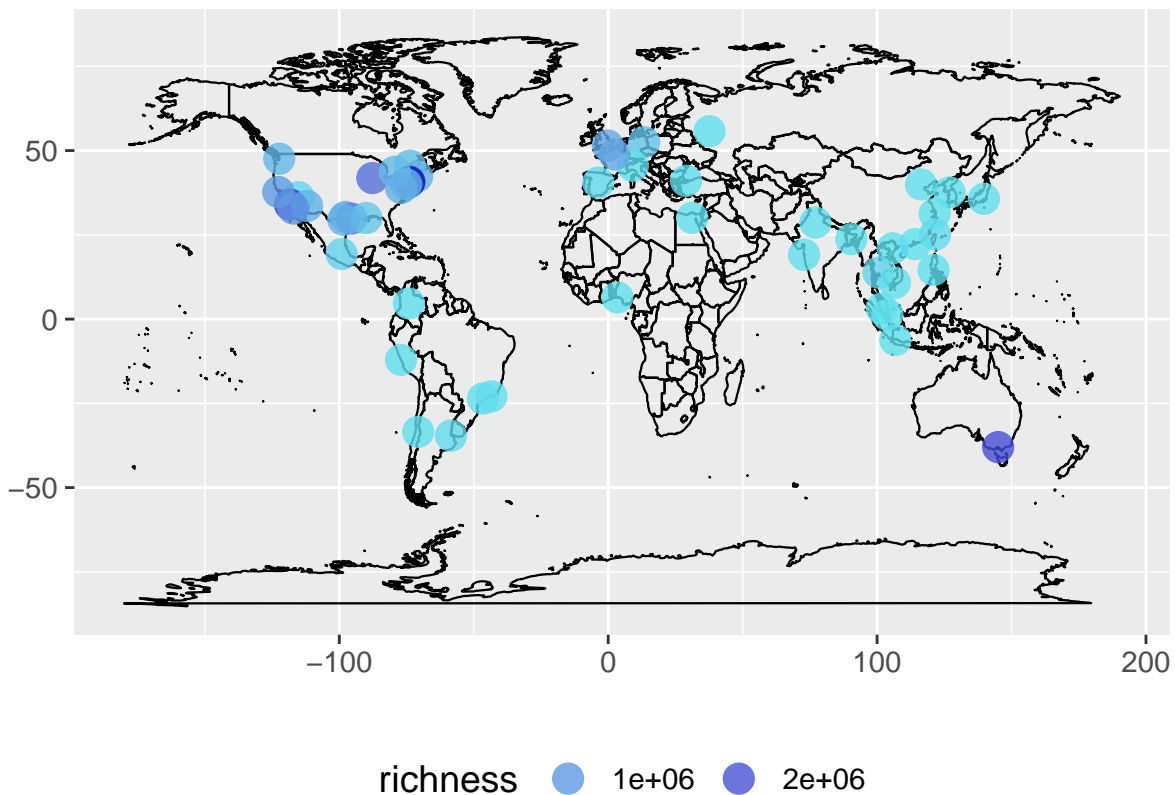
```
out <- merge(dat, ldply(out), by = "city")
```

302 *Add centroids from bounding boxes*

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

303 *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))
```



304

305 *Valley oak occurrence data comparison*

306 This example comes from [Antonio J. Perez-Luque](#) who [shared his plot on Twitter](#). Antonio compared
307 the occurrences of Valley Oak (*Quercus lobata*) from [GBIF](#) to the distribution of the same species from
308 the [Atlas of US Trees](#).

309 *Load libraries*

```
library('rgbif')
library('raster')
library('sp')
library('maptools')
library('rgeos')
library('scales')
```

310 *Get GBIF Data for Quercus lobata*

```
keyQ1 <- name_backbone(name='Quercus lobata', kingdom='plants')$speciesKey
dat.Q1 <- occ_search(taxonKey=keyQ1, return='data', limit=50000)
```

311 *Get Distribution map of Q. lobata Atlas of US Trees (Little, E.)*

312 From <http://esp.cr.usgs.gov/data/little>. And save shapefile in same directory

```
url <- 'http://esp.cr.usgs.gov/data/little/querloba.zip'
tmp <- tempdir()
download.file(url, destfile = "~/querloba.zip")
unzip("~/querloba.zip", exdir = tmp)
ql <- readShapePoly(file.path(tmp, "querloba.shp"))
```

313 *Get Elevation data of US*

```
alt.USA <- getData('alt', country = 'USA')
```

314 *Create Hillshade of US*


```

alt.USA <- alt.USA[[1]]
slope.USA <- terrain(alt.USA, opt = 'slope')
aspect.USA <- terrain(alt.USA, opt = 'aspect')
hill.USA <- hillShade(slope.USA, aspect.USA, angle = 45, direction = 315)

```

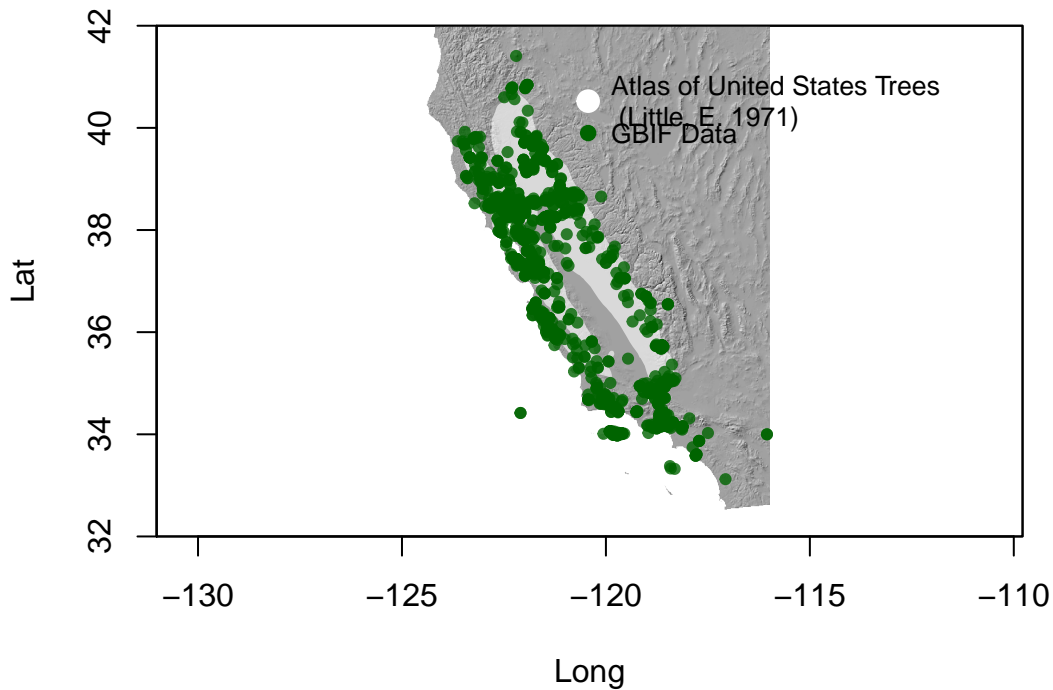
315 *Plot map*

```

plot(hill.USA, col = grey(0:100/100), legend = FALSE,
     xlim = c(-125, -116), ylim = c(32, 42),
     main = 'Distribution of Quercus lobata',
     xlab = "Long", ylab = 'Lat')
# add shape from Atlas of US Trees
plot(ql, add = TRUE, col = alpha("white", 0.6),
     border = FALSE)
# add Gbif presence points
points(dat.Ql$decimalLongitude, dat.Ql$decimalLatitude,
       cex = .7, pch = 19, col = alpha("darkgreen", 0.8))
legend(x = -121, y = 40.5, "GBIF Data", pch = 19,
       col = 'darkgreen', bty = 'n', pt.cex = 1, cex = .8)
legend(x = -121, y = 41.5,
       legend = "Atlas of United States Trees \n (Little, E. 1971)",
       pt.cex = 1.5, cex = .8, pch = 19, col = 'white', bty = 'n')

```

Distribution of *Quercus lobata*



316

317 Conclusions and future directions

318 The `rgbif` and `pygbif` libraries provide programmatic interfaces to GBIF's application programming
 319 interface (API) - a powerful tool for working with species occurrence data, and facilitating reproducible
 320 research. In fact, the `rgbif` package has already been used in 22 scholarly publications (as of 2016-08-10)
 321 (Amano, Lamming & Sutherland, 2016, Bartomeus et al. (2013), Barve (2014), Bone et al. (2015),
 322 Collins et al. (2015), Drozd & Šipoš (2013), Kong, Huang & Duan (2015), Richardson, Roux & Wilson
 323 (2015), Turner, Fréville & Rieseberg (2015), Verheijen et al. (2015), Zizka & Antonelli (2015), Butterfield
 324 et al. (2016), Dellinger et al. (2015), Feitosa et al. (2015), Malhado et al. (2015), Werner et al. (2015),
 325 Robertson, Visser & Hui (2016), Davison et al. (2015), Janssens et al. (2016)).

326 The `rgbif` package is relatively stable, and should not have many breaking changes unless necessitated
 327 due to changes in the GBIF API.

328 The `pygbif` library is in early development, and will greatly benefit from any feedback and use cases.

329 One area of focus in the future is to attempt to solve many use cases that have been brought up with

330 respect to GBIF data. For example, some specimens are included in GBIF that are located in botanical
331 gardens. For many research questions, researchers are interested in “wild” type occurrences, not those
332 in human curated scenarios. Making removal of these occurrences easy would be very useful, but is
333 actually quite a hard problem. There are many other problems like this.

334 Acknowledgments

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

337 Data Accessibility

338 All scripts and data used in this paper can be found in the permanent data archive Zenodo under
339 the digital object identifier (DOI). This DOI corresponds to a snapshot of the GitHub repository at
340 <https://github.com/sckott/msrgbif>. Software can be found at <https://github.com/ropensci/rgbif> and
341 <https://github.com/sckott/pygbif>, both under MIT licenses. We thank all the users that have used
342 `rgbif` and `pygbif` and have given feedback and reported bugs. In addition, we greatly appreciate the
343 all contributions to the two packages, found at <https://github.com/ropensci/rgbif/graphs/contributors>
344 and <https://github.com/sckott/pygbif/graphs/contributors>.

345 References

- 346 Amano T., Lamming JDL., Sutherland WJ. 2016. Spatial gaps in global biodiversity information and
347 the role of citizen science. *BioScience* 66:393–400.
- 348 Bartomeus I., Park MG., Gibbs J., Danforth BN., Lakso AN., Winfree R. 2013. Biodiversity ensures
349 plant-pollinator phenological synchrony against climate change. *Ecology Letters* 16:1331–1338.
- 350 Barve V. 2014. Discovering and developing primary biodiversity data from social networking sites: A
351 novel approach. *Ecological Informatics* 24:194–199.
- 352 Beck J., Ballesteros-Mejia L., Buchmann CM., Dengler J., Fritz SA., Gruber B., Hof C., Jansen
353 F., Knapp S., Kreft H., Schneider A-K., Winter M., Dormann CF. 2012. Whats on the horizon for
354 macroecology? *Ecography* 35:673–683.

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

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

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

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

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

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

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

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

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

376 Dellinger AS., Essl F., Hojsgaard D., Kirchheimer B., Klatt S., Dawson W., Pergl J., Pyšek P., Kleunen
377 M van., Weber E., Winter M., Hörandl E., Dullinger S. 2015. Niche dynamics of alien species do not
378 differ among sexual and apomictic flowering plants. *New Phytologist* 209:1313–1323.

379 Drozd P., Šipoš J. 2013. R for all (i): Introduction to the new age of biological analyses. *Casopis*
380 *slezskeho zemskeho muzea (A)* 62.

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

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

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

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

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

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

394 Janssens SB., Vandelook F., Langhe ED., Verstraete B., Smets E., Vandenhoutte I., Swennen R. 2016.
395 Evolutionary dynamics and biogeography of musaceae reveal a correlation between the diversification
396 of the banana family and the geological and climatic history of southeast asia. *New Phytologist*
397 210:1453–1465.

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

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

402 María Mendoza., Ospina OE., Cárdenas-Henao H., García-R JC. 2015. A likelihood inference of
403 historical biogeography in the world’s most diverse terrestrial vertebrate genus: Diversification of
404 direct-developing frogs (craugastoridae: Pristimantis) across the neotropics. *Molecular Phylogenetics*
405 *and Evolution* 85:50–58.

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

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

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

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

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

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

420 Werner G.D.A., Cornwell W.K., Cornelissen J.H.C., Kiers E.T. 2015. Evolutionary signals of symbiotic
 421 persistence in the legume-rhizobia mutualism. *Proceedings of the National Academy of Sciences* 112:10262–
 422 10269.

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