Material suplementario de:

Complementando inventarios biológicos con datos abiertos a través de un método semiautomatizado

Publicado en:
Revista de Biología Tropical

Sergio A. Cabrera-Cruz, José Luis Aguilar-López, Rafael Villegas Patraca

2024-02-16

Índice

Resumen

Este documento acompaña al artículo titulado "Complementando inventarios biológicos con datos abiertos a través de un método semiautomatizado", de Sergio A. Cabrera-Cruz, José Luis Aguilar-López y Rafael Villegas Patraca, publicado en la Revista de Biología Tropical. Contiene código de R necesario para generar listados de especies usando datos reportados en la plataforma GBIF (Global Biodiversity Information Facility). El código puede ser ajustado para descargar registros de biodiversidad de cualquier área y grupo biológico. Si usas el código, por favor cita nuestro artículo, así como a B. R. Ribeiro et al. (2022), sobre cuyo trabajo descansa el nuestro.

A manera de ejemplo, se utilizan coordenadas de un AICA (Área de Importancia para la Conservación de las Aves) para descargar desde GBIF registros de aves reportadas en su interior, generando un listado final de especies. Sin embargo, se proporcionan maneras de obtener las coordenadas de tu propia área de interés considerando tres escenarios posibles: 1) que sólo tengas un par de coordenadas alrededor de las cuales se definirá un polígono con radio de tu elección, 2) que sólo tengas dos pares de coordenadas definiendo el área mínima de interés, y 3) que tengas un archivo vectorial con el polígono del área de interés (e.g. shapefile o geopackage). El tiempo de ejecución de la descarga de datos será afectado tanto por el tamaño de tu área de interés como por la cantidad de registros disponibles en GBIF para tu grupo biológico de interés.

El documento tiene cuatro secciones principales:

1. Declara variables (como las coordenadas de tu área de interés y tus credenciales de GBIF)
2. Descarga datos de GBIF
3. Limpia los datos descargados
4. Genera tu listado final

Para poder utilizar este documento es recomendable estar familiarizado con R, pero no es necesario ser experto/a. Necesitas una versión reciente de R (https://cran.r-project.org/), RStudio (https://posit.co/download/rstudio-desktop/), y contar con conexión a internet. Se recomienda estar registrado/a en GBIF (i.e. contar con un nombre de usuario y contraseña) pues eso simplifica el proceso de descarga de datos y además la descarga genera una referencia citable. Puedes registrarte en la página principal de GBIF (https://www.gbif.org/es/) dando click al botón "Iniciar sesión", eligiendo después la opción "Registrarse". Sin embargo, se incluye una opción para descargar datos sin haberte registrado a GBIF (NO RECOMENDADO).

NOTA: Copia y pega a un script de R sólo las partes de código y texto que están dentro de los recuadros grises. Si quieres agregar más texto a tu script en R deberás agregar el símbolo # antes de cada línea.

Instala paquetes y/o carga paquetería necesaria

NOTA: Esto podría tomar unos minutos.

Primero instala manualmente los paquetes devtools() y rnaturalearthhires().

```
# Instala devtools() y rnaturalearthhires()
utils::install.packages("devtools")
devtools::install_github("ropensci/rnaturalearthhires")
```

Ahora instala y/o carga toda la paquetería necesaria.

```r
# Instala paquetería necesaria

# Paquetes necesarios
paquetes <- c("rgbif", "sf",
              "spocc", "mapview",
              "bdc", "dplyr",
              "rnaturalearthdata",
              "rnaturalearthhires",
              "devtools", "doParallel",
              "leaflet", "units")


# Verifica si están instalados; de lo contrario, los instala
inst <- paquetes %in% installed.packages()


if(length(paquetes[!inst]) > 0){
  install.packages(paquetes[!inst])}


# Carga los paquetes
lapply(paquetes,
       require,
       character.only = TRUE)


# Limpia un poco
rm(list = ls())
```

Establece directorio de trabajo

IMPORTANTE: A lo largo de este código se incluyen comandos para guardar en tu computadora archivos intermedios y para leerlos subsecuentemente. Los comandos asumen que declaraste aquí la dirección completa de la carpeta donde quieres que se guarden los datos que vas a generar.

```r
# Declara aquí la dirección completa de la carpeta donde quieres que se guarden los
# archivos y datos que se van a generar
mi_directorio <- "guardar/mis_datos/en_esta_carpeta/"

# NOTA:
# Usuari@s de Windows: No uses diagonales inversas (\) sino diagonales regulares (/)
# Usuari@s de Mac: Tal vez debas agregar el símbolo "~" al inicio de tu directorio
```

Declara variables

En esta sección declara algunas variables para poder correr el código.

Datos de usuario de GBIF

En primer lugar, declara tu nombre de usuario, contraseña, y correo electrónico con los que creaste tu cuenta en GBIF. Si no te registraste a GBIF, puedes omitir este paso (NO RECOMENDADO)

```
# Declara tus datos de usuario de GBIF

# NOTA: Reemplaza la información con tu nombre de usuario, contraseña y correo electrónico
# registrados en GBIF.

mi_usuario <- "mi_nombre_de_usuario_en_GBIF"
mi_pass  <- "mi_contraseña_en_GBIF"
mi_email <- "mi_correo_registrado_en_GBIF"
```

Grupo biológico de interés

Ahora, declara tu grupo biológico (taxón) de interés. GBIF asigna códigos numéricos únicos a cada taxón. La función name_backbone() del paquete rgbif() nos ayuda a encontrar el código correspondiente a nuestro grupo de interés. El trabajo al que acompaña este documento fue sobre Aves, pero en las líneas de abajo puedes reemplazar el nombre de la Clase taxonómica por la que sea de tu interés (e.g. Reptilia, Mammalia, Amphibia). Para el caso de plantas, usa "Plantae" tanto en 'mi_grupo' como en el argumento "kingdom" dentro de la función rgbif::name_backbone().

```
# Obtén clave del taxón que sea de tu interés. Puedes reemplazar "Aves" por nombres
# como "Amphibia", "Reptilia", "Plantae", etc.

mi_grupo <- "Aves"
mi_grupo <- rgbif::name_backbone(name = mi_grupo,
                                 kingdom = "animals")

mi_grupo <- mi_grupo$usageKey
```

Coordenadas del área de interés

Las coordenadas deben estar en formato WKT ("Well Known Text").

Como ejemplo, en este documento usaremos las siguientes coordenadas pero más adelante se muestra cómo generar tus propias coordenadas en formato WKT para tu propia área de interés

```
# Estas coordenadas son del polígono simplificado del AICA Omiltemi
# (http://avesmx.conabio.gob.mx/FichaRegion.html#AICA_21). Fueron generadas con
# el código disponible más adelante en el punto "Caso 3: Con archivo vectorial" y con
# un shapefile descargado de
# http://www.conabio.gob.mx/informacion/gis/?vns=gis_root/region/biotic/aicas15gw

coords_wkt <- paste0("POLYGON ((-99.69341 17.50438, -99.65889 17.52621,",
                             "-99.66122 17.53457, -99.64991 17.55098,",
                             "-99.65889 17.56104, -99.66477 17.59045,",
                             "-99.67948 17.59045, -99.68784 17.59742,",
                             "-99.69295 17.58503, -99.70271 17.58318,",
                             "-99.70611 17.56739, -99.72515 17.56243,",
                             "-99.7363 17.54695, -99.72965 17.54757,",
```

```
                                "-99.7188 17.53581, -99.72515 17.51986,",
                                "-99.72377 17.51434, -99.70441 17.53674,",
                                "-99.69791 17.50748, -99.69341 17.50438))")


# Genera un área para que puedas visualizar el polígono
mi_area <- sf::st_as_sf(sf::st_as_sf(data.frame(coords_wkt),
                                     wkt = 1,
                                     crs = 4326))


sf::st_geometry(mi_area) <- "geometry"
```

Visualiza el área de interés

OPCIONAL: Visualiza tu área de interés en un mapa interactivo. La función de abajo abre una ventana de tu navegador de internet.

```
# Visualiza tu área de interés

# Opción 1: WKT directo
spocc::wkt_vis(coords_wkt,
               zoom = 10,
               maptype = "terrain",
               browse = TRUE)
```

Descarga datos de GBIF

El archivo que se va a descargar contiene todos los registros de ocurrencias de las especies pertenecientes al grupo biológico indicado en el argumento 'mi_grupo' que han sido observadas y reportadas históricamente a GBIF. Por tanto, el archivo puede contener múltiples registros para una misma especie que hayan sido reportadas en distintos años e incluso bajo distintos nombres. Todo esto se atenderá en la siguiente sección "Limpia registros con bdc()".

Los argumentos especificados en la función rgbif::occ_download() van a usar tus datos de usuario en GBIF para descargar:

- Todos los registros de tu grupo biológico de interés (aves en este ejemplo)
- Registros están dentro del polígono definido por las coordenadas en mi_area_wkt()
- Sólo registros con coordenadas
- Ignorando registros con datos geoespaciales problemáticos
- Manteniendo sólo registros catalogados como "presentes" en el área
- Removiendo fósiles
- Descargando los datos en formato CSV (que también puedes abrir en Excel)

```r
# Descarga datos de GBIF

# Especifica el año a partir del cuál te interesan registros de GBIF
mi_año <- 1900


# Solicita la descarga de datos
occ_down <- rgbif::occ_download(user = mi_usuario,
                                pwd = mi_pass,
                                email = mi_email,
                                rgbif::pred("taxonKey", mi_grupo),
                                rgbif::pred_within(coords_wkt),
                                rgbif::pred_gte("year", mi_año),
                                rgbif::pred("hasCoordinate", TRUE),
                                rgbif::pred("hasGeospatialIssue", FALSE),
                                rgbif::pred("occurrenceStatus", "PRESENT"),
                                rgbif::pred_not(rgbif::pred_in("basisOfRecord",
                                                               c("FOSSIL_SPECIMEN"))),
                                format = "SIMPLE_CSV")


# Obtén la clave de la descarga
down_key <- names(split(occ_down,
                        occ_down))


# Espera unos minutos y después revisa el estado de la descarga
rgbif::occ_download_wait(down_key)


# Cuando haya terminado, descarga a tu computadora el archivo generado
mis_datos <- rgbif::occ_download_get(key = down_key,
                                     path = mi_directorio)


# Lee los datos producto de la descarga
mis_datos <- rgbif::occ_download_import(key = down_key,
                                        path = mi_directorio)


# Exporta lista de especies en formato .CSV (este formato se puede abrir con Excel)
write.csv(mis_datos,
          file = paste0(mi_directorio,
                        "01_lista_de_especies_GBIF.csv"),
          row.names = F)


# Lee los metadatos asociados con la descarga
down_meta <- rgbif::occ_download_meta(down_key)


# IMPORTANTE: Toma nota de estos datos, serán útiles para futura referencia
### Cita de la descarga
```

```
rgbif::gbif_citation(down_key)$download

### DOI (Digital Object Identifier) de la descarga
down_meta$doi

### Clave de la descarga
down_meta$key

### Enlace (URL) de la descarga
down_meta$downloadLink

### Fecha de descarga
down_meta$created

### Número de registros descargados (NOTA: no es lo mismo que número de especies)
down_meta$totalRecords
```

Visualiza los registros descargados en un mapa simple

```
# Visualiza los registros descargados de GBIF

# Genera un objeto espacial
mis_datos_figura <- sf::st_as_sf(mis_datos[, c("scientificName",
                                               "decimalLongitude",
                                               "decimalLatitude")],
                                 coords = c("decimalLongitude",
                                            "decimalLatitude"))

par(mfrow = c(1,1))
plot(mi_area$geometry, col = "lightgray")
plot(mis_datos_figura$geometry, add = T)
```

Limpia registros con bdc()

Una vez que hayas descargado los datos de GBIF, es necesario limpiarlos para eliminar o corregir registros erróneos o sospechosos. La limpieza consiste en 4 pasos: prefiltrado, validación taxonómica, validación espacial, y validación temporal. En el artículo se describe con mayor detalle la razón por la cual hay que dar estos pasos y los posibles errores en los datos descargados.

IMPORTANTE: Las siguientes secciones (Prefiltrado, Revisión de la taxonomía, Validación espacial, y Validación temporal) deben ser ejecutadas de manera secuencial.

NOTA Y RECONOCIMIENTO Toda esta sección de limpieza está basada en el trabajo de B. R. Ribeiro et al. (2022). Los cuatro pasos fueron desarrollados por esos autores como parte de su paquete bdc(), y mucho del código en esta sección es una adaptación del suyo. Puedes explorar más detalles del protocolo de limpieza visitando la página https://brunobrr.github.io/bdc/index.html.

```
# Preparación para limpiar datos descargados de GBIF

# 1. Lee de vuelta los datos descargados de GBIF.
mis_datos <- read.csv(paste0(mi_directorio,
```

```
                              "01_lista_de_especies_GBIF.csv"))


# 2. Agrega una columna llamada "database_id" y llénala con el nombre que te sirva. Esta
#    columna es necesaria para que se ejecute una función más adelante.
mis_datos$database_id <- "mi_area_de_estudio"


# 3. Quédate sólo con las columnas que las funciones del paquete bdc() necesita o sugiere
#    para limpiar la base de datos.

# Estos son los nombres de las columnas necesarias y sugeridas para bdc()
bdc_columnas <- c("datasetName", "fileName", "scientificName",      # NECESARIAS
                  "decimalLatitude", "decimalLongitude",            # NECESARIAS
                  "database_id",                                    # NECESARIAS
                  "occurrenceID", "basisOfRecord",                  # SUGERIDAS
                  "verbatimEventDate",  "eventDate", "country",     # SUGERIDAS
                  "stateProvince", "county", "locality",            # SUGERIDAS
                  "identifiedBy", "coordinateUncertaintyInMeters", # ADICIONALES
                  "coordinatePrecision", "recordedBy")              # ADICIONALES


# 4. Filtra tu base de datos descargada de GBIF, tomando sólo las columnas que bdc()
#    necesita o sugiere.
mis_datos <- mis_datos[, colnames(mis_datos) %in% c(bdc_columnas)]


# OPCIONAL: ¿Cuáles de las columnas con las que te quedaste están en la lista de las
#           necesarias o sugeridas por bdc()?
setdiff(bdc_columnas, colnames(mis_datos))
```

Prefiltrado

Este paso incluye varias revisiones de posibles errores en los registros descargados de GBIF.

Para más detalles, consulta la información oficial en la cual nos basamos, disponible en https://cran.r-project.org/web/packages/bdc/vignettes/prefilter.html

```
# Limpieza de registros descargados. Paso 1: Prefiltrado

# 1. Identifica registros sin nombre científico
prefiltrado <- bdc::bdc_scientificName_empty(data = mis_datos,
                                             sci_name = "scientificName")

# NOTA: Esta y las siguientes funciones generan notificaciones avisando el resultado del
#       proceso. Mientras no aparezca la palabra 'ERROR', todo está bien


# 2. Identifica registros con información geográfica parcial
prefiltrado <- bdc::bdc_coordinates_empty(data = prefiltrado,
                                          lat = "decimalLatitude",
                                          lon = "decimalLongitude")
```

```r
# 3. Identifica registros con coordenadas geográficas erróneas (con latitud > 90 o -90 y
#    longitud >180 o -180).
prefiltrado <- bdc::bdc_coordinates_outOfRange(data = prefiltrado,
                                               lat = "decimalLatitude",
                                               lon = "decimalLongitude")


# 4. Identifica registros de fuentes que pudieran no ser confiables, como dibujos,
#    fotografías, registros fósiles, entre otros.
prefiltrado <- bdc::bdc_basisOfRecords_notStandard(data = prefiltrado,
                                                   basisOfRecord = "basisOfRecord",
                                                   names_to_keep = "all")


# 5. Genera nombre de país usando las coordenadas geográficas. Esto funciona como un
#    filtro extra para asegurarse de que los registros sí están asociados al país en el
#    que esté tu polígono o área de interés.
#    NOTA: La función bdc_country_from_coordinates() depende de funciones en otros
#          paquetes, uno de ellos es sf(). Ejecuta esta línea para evitar un error que
#          surge cuando sf() intenta usar geometría esférica, que no queremos
sf::sf_use_s2(FALSE)


prefiltrado <- bdc::bdc_country_from_coordinates(data = prefiltrado,
                                                 lat = "decimalLatitude",
                                                 lon = "decimalLongitude",
                                                 country = "country")


# 6. Estandariza el nombre del país.
prefiltrado <- bdc::bdc_country_standardized(data = prefiltrado,
                                             country = "country")


# 7. Corrige latitudes y longitudes que pudieran estar transpuestas.
#    NOTA: El argumento 'border_buffer' está dado en grados decimales (border_buffer = 0.2
#          es aproximadamente 22 km en el ecuador).
#    NOTA: Es posible que esta operación arroje una advertencia (Warning) porque la
#          función bdc_coordinates_transposed() en la versión 1.1.4 del paquete bdc()
#          utiliza una versión desactualizada de la función filter() del paquete dplyr()
#          [checada por última vez el 20 de Mayo de 2023].
prefiltrado <- bdc::bdc_coordinates_transposed(data = prefiltrado,
                                               id = "database_id",
                                               sci_names = "scientificName",
                                               lat = "decimalLatitude",
                                               lon = "decimalLongitude",
                                               country = "country_suggested",
                                               countryCode = "countryCode",
                                               border_buffer = 0.2,
                                               save_outputs = FALSE)


# 8. Identifica registros fuera del área del interés
```

```
#    NOTA: El argumento dist() está dado en grados decimales (dist = 0.1 son
#          aproximadamente 11 km en el ecuador).
prefiltrado <- bdc::bdc_coordinates_country_inconsistent(data = prefiltrado,
                                                         country_name = "Mexico",
                                                         country = "country_suggested",
                                                         lon = "decimalLongitude",
                                                         lat = "decimalLatitude",
                                                         dist = 0.1)


# 9. Genera una columna llamada ".summary", la cual suma los resultados de todas las
#    pruebas de validación. Esta columna indica FALSO en los registros que hayan fallado
#    aunque sea una sola de las pruebas de validación, lo que los hace potencialmente
#    inválidos o al menos sospechosos.
prefiltrado <- bdc::bdc_summary_col(data = prefiltrado)


# 10. Elimina registros potencialmente inválidos o sospechosos y las columnas con las
#     pruebas de validación.
salida <-
  prefiltrado %>%
  dplyr::filter(.summary == TRUE) %>%
  bdc::bdc_filter_out_flags(data = .,
                            col_to_remove = "all")


# 11. Guarda la base de datos generada
write.csv(salida,
          file = paste0(mi_directorio,
                        "02_mi_lista_de_especies_prefiltrada.csv"),
          row.names = F)
```

Revisión de la taxonomía

Este paso compara los nombres científicos en los datos descargados de GBIF contra una base taxonómica moderna de tu elección, corrigiendo (hasta cierto punto) nombres desactualizados, sinonimias, y otros casos.

Para más detalles, consulta la información oficial en la cual nos basamos, disponible en https://cran.r-project.org/web/packages/bdc/vignettes/taxonomy.html

```
# Limpieza de registros descargados. Paso 2: Revisión de la taxonomía


# 1. Lee de vuelta los datos prefiltrados.
mis_datos <- read.csv(paste0(mi_directorio,
                             "02_mi_lista_de_especies_prefiltrada.csv"))


# 2. Declara algunas de las variables necesarias para la función bdc_query_names_taxadb()
#    que se usará un poco más adelante.

### Nombre del rango taxonómico de interés, en inglés (opciones: "kingdom", "phylum",
#   "class", "order", "family" o "genus")
```

```
rank = "class"

### Grupo biológico de interés. Debe ser acorde con el rango taxonómico definido arriba.
rank_name = "Aves"

### Si la función encuentra sinonimias en los nombres, ¿reemplazarlos por nombres
#   aceptados? (opciones: TRUE o FALSE).
replace_synonyms = TRUE

### Si algún nombre está mal escrito (tiene un error ortográfico) ¿Quieres que la función
#   intente un encontrar nombre alternativo? (opciones: TRUE o FALSE).
suggest_names = TRUE

### Distancia ortográfica entre los nombres en tu base de datos y los que la función puede
#   buscar como alternativa (valor entre 0 y 1)
suggestion_distance = 0.9

### ¿Qué base de datos taxonómica quieres que la función consulte? (opciones: "gbif",
#   "itis", "ncbi", "col", "tpl", "fb", "slb", "wd", "ott", "iucn")
#   Para más detalles sobre las opciones, consulta ?bdc::bdc_query_names_taxadb
db = "gbif"


# 3. Limpia nombres científicos (revisa posibles errores en los registros de GBIF)
analiza_nombres <- bdc::bdc_clean_names(sci_names = mis_datos$scientificName,
                                        save_outputs = FALSE)


# 4. Combina los nombres revisados con la base completa
analiza_nombres <- analiza_nombres %>%
  dplyr::select(.uncer_terms,
                names_clean)

mis_datos <- dplyr::bind_cols(mis_datos, analiza_nombres)


# 5. Armonización de nombres.
# NOTA: La primera vez que uses esta función, descargará a tu computadora la base de datos
#       especificada en el argumento 'db', con la finalidad de que esté disponible de
#       manera local en siguientes ocasiones que la uses. Por tanto, esta función tardará
#       más la primera vez que la ejecutes.
consulta_nombres <- bdc::bdc_query_names_taxadb(sci_name = mis_datos$names_clean,
                                                rank_name = rank_name,
                                                rank = rank,
                                                replace_synonyms = replace_synonyms,
                                                suggest_names = suggest_names,
                                                suggestion_distance = suggestion_distance,
                                                db = db)


# 6. Combina el resultado de la armonización de nombres con la base original
mis_datos <-
  mis_datos %>%
```

```
  dplyr::rename(verbatim_scientificName = scientificName) %>%
  dplyr::select(-names_clean) %>%
  dplyr::bind_cols(., consulta_nombres)


# 7. Nombres sin resolver. Es muy posible que bdc::bdc_query_names_taxadb() no haya
#    podido identificar todos los nombres científicos en tu base de datos. Las siguientes
#    líneas generan un objeto con los registros de especies cuyos nombres no fueron
#    identificados.
sin_resolver <- bdc::bdc_filter_out_names(data = mis_datos,
                                          col_name = "notes",
                                          taxonomic_status = "accepted",
                                          opposite = TRUE)


sin_resolver <- sin_resolver[, c("original_search",
                                 "notes")]


# 8. Elimina duplicados de los registros con nombres sin resolver y elimina de tus datos
#    los registros de especies con nombres científicos sin resolver.
sin_resolver <- unique(sin_resolver)

mis_datos <- mis_datos[!mis_datos$original_search %in% sin_resolver$original_search, ]


# IMPORTANTE:
# Es necesario verificar manualmente los nombres sin resolver, para decidir si dejar esas
# especies fuera de tu listado final o si las puedes regresar.


# 9. Exporta / guarda tu base con la taxonomía aceptada.
write.csv(mis_datos,
          file = paste0(mi_directorio,
                        "03_mi_lista_de_especies_nombres_limpios.csv"),
          row.names = F)


# OPCIONAL: Guarda la base de especies con nombres por resolver
# NOTA: La función if() que encierra todo se asegura de que las líneas en su interior se
# ejecuten sólo si el objeto sin_resolver() contiene algún dato.
if(nrow(sin_resolver) > 0){
  write.csv(sin_resolver,
            file = paste0(mi_directorio,
                          "04_especies_sin_resolver.csv"),
            row.names = F)
  }
```

Validación espacial

Identifica registros con coordenadas geográficas que tengan posibles errores.

Para más detalles, consulta la información oficial en la cual nos basamos, disponible en https://cran.r-

project.org/web/packages/bdc/vignettes/space.html

```
# Limpieza de registros descargados. Paso 3: Validación espacial


# 1. Lee la base de datos con la taxonomía revisada
mis_datos <- read.csv(paste0(mi_directorio,
                             "03_mi_lista_de_especies_nombres_limpios.csv"))


# 2. Identifica registros con problemas espaciales comunes
val_espacial <- bdc::bdc_coordinates_precision(data = mis_datos,
                                               lon = "decimalLongitude",
                                               lat = "decimalLatitude",
                                               ndec = c(0, 1))


# 3. Identifica registros con posibles problemas más específicos, usando el paquete
#    CoordinateCleaner()

#    NOTA: Es posible que esta función te arroje algunas advertencias (Warnings) pero
#          debería ejecutarse correctamente y el resultado ser válido. Sin embargo,
#          una advertencia a tener en cuenta es que la función no evalúa especies con
#          menos de 7 registros en tu base de datos.
val_espacial <- CoordinateCleaner::clean_coordinates(x = val_espacial,
                                                     lon = "decimalLongitude",
                                                     lat = "decimalLatitude",
                                                     species = "scientificName",
                                                     countries = "country",
                                                     capitals_rad = 2000,
                                                     centroids_rad = 1000,
                                                     centroids_detail = "both",
                                                     inst_rad = 100,
                                                     outliers_method = "quantile",
                                                     outliers_mtp = 5,
                                                     outliers_td = 1000,
                                                     outliers_size = 10,
                                                     range_rad = 0,
                                                     zeros_rad = 0.5,
                                                     country_refcol = "countryCode",
                                                     value = "clean")

# Para detalles de lo que se trata cada argumento y de los muchos otros que puedes
# especificar y controlar, consulta ?CoordinateCleaner::clean_coordinates().



# 4. Genera columna llamada ".summary" la cual indica si algún registro falló alguna de
#    las pruebas espaciales.
val_espacial <- bdc::bdc_summary_col(data = val_espacial)


# 5. Filtra la base
salida <-
```

```
  val_espacial %>%
  dplyr::filter(.summary == TRUE) %>%
  bdc::bdc_filter_out_flags(data = ., col_to_remove = "all")


# 6. Exporta / guarda
write.csv(salida,
          file = paste0(mi_directorio,
                        "05_mi_lista_de_especies_validacion_espacial.csv"),
          row.names = F)
```

Validación temporal

Extrae el año de colecta o de observación del registro e identifica años de colecta dudosos, erróneos o no proporcionados.

Para más detalles, consulta la información oficial en la cual nos basamos, disponible en: https://cran.r-project.org/web/packages/bdc/vignettes/time.html

```
# Limpieza de registros descargados. Paso 4: Validación temporal


# 1. Lee los datos validados espacialmente
mis_datos <- read.csv(paste0(mi_directorio,
                             "05_mi_lista_de_especies_validacion_espacial.csv"))


# 2. Identifica registros sin año de observación
#    NOTA: El argumento 'eventDate' puede funcionar con el valor 'eventDate' o
#          'verbatimEventDate'. Utiliza el que venga en tu base de datos
val_temporal <- bdc::bdc_eventDate_empty(data = mis_datos,
                                         eventDate = "eventDate")


# 3. Extrae el año a partir de la columna 'eventDate' o 'verbatimEventDate'
#    NOTA: El argumento 'eventDate' puede funcionar con el valor 'eventDate' o
#          'verbatimEventDate'. Utiliza el que venga en tu base de datos
val_temporal <- bdc::bdc_year_from_eventDate(data = val_temporal,
                                             eventDate = "eventDate")


# 4. Identifica registros colectados en años fuera del umbral de interés
val_temporal <- bdc::bdc_year_outOfRange(data = val_temporal,
                                         eventDate = "year",
                                         year_threshold = 1900)


# 5. Genera columna llamada ".summary" identificando registros con posibles problemas
val_temporal <- bdc::bdc_summary_col(data = val_temporal)


# 6. Exporta / guarda.
write.csv(val_temporal,
```

```
          file = paste0(mi_directorio,
                        "06_mi_lista_de_especies_validacion_temporal.csv"),
          row.names = F)
```

Filtrado final

Elimina registros potencialmente erróneos o sospechosos que hayan sido identificados en cada una de las evaluaciones y genera una base de datos "limpia".

```
# Filtrado final


# 1. Lee de vuelta los datos validados temporalmente.
mis_datos <- read.csv(paste0(mi_directorio,
                             "06_mi_lista_de_especies_validacion_temporal.csv"))


# 2. Filtra registros erróneos o sospechosos.
mis_datos <-
  mis_datos %>%
  bdc::bdc_filter_out_flags(data = .,
                            col_to_remove = "all")


# 3. Toma sólo las especies aceptadas después de aplicar los filtros y las columnas con
#    información taxonómica.
mi_lista <- mis_datos[grepl("accepted",
                            mis_datos$notes),
                      c("kingdom", "phylum", "class", "order", "family",
                        "genus", "specificEpithet", "infraspecificEpithet",
                        "scientificName")]


# 4. Quédate sólo con las especies únicas.
mi_lista <- unique(mi_lista)


# OPCIONAL: ¿Cuántas especies hay en 'mi_lista'?
length(unique(mi_lista$scientificName))


# OPCIONAL: ¿Cuántos registros están duplicados? (en teoría, cero)
sum(duplicated(mi_lista$scientificName))


# OPCIONAL: Cambia el nombre de las columnas de inglés a español.
colnames(mi_lista) <- c("Reino", "Phylum", "Clase", "Orden", "Familia",
                        "Género", "Epiteto específico",
                        "Epiteto infraespecífico",
                        "Nombre científico")
```

Guarda tu listado de especies

```
# Guarda tu lista en formato .csv, formato que también se puede abrir en Excel
write.csv(mi_lista,
          file = paste0(mi_directorio,
                        "07_mi_lista_especies_unicas.csv"),
          row.names = F)


# OPCIONAL: Abre aquí mismo el listado para que veas lo que generaste
View(mi_lista)
```

Listo. Éste último archivo contiene la lista de especies reportadas en GBIF para tu área de interés. Tu listado debería estar ubicado aquí:

```
# Directorio y archivo con mi listado de especies final
paste0(mi_directorio,
       "07_mi_lista_especies_unicas.csv")
```

Tu propia área de interés

Genera tus propias coordenadas en formato WKT

A continuación, se muestran tres maneras de generar coordenadas en formato WKT para tu propia área de interés.

Caso 1: Con un sólo punto

Si tienes un sólo punto entonces es necesario crear un área alrededor de esa ubicación.

El par de coordenadas en el ejemplo de abajo fueron extraídas usando Google Earth y aproximan la ubicación del área de estudio de Martínez-Morales et al. (2013).

```
# Genera coordenadas WKT de un polígono alrededor de tu punto de interés

# IMPORTANTE: Primero la longitud, después la latitud
mi_punto <- c(-98.50381, 20.13812)


# Genera un punto espacial
mi_punto <- sf::st_point(mi_punto, dim = "XY")


# Asigna proyección geográfica
mi_punto <- sf::st_sfc(mi_punto, crs = 4326)


# Proyecta a un sistema de referencia que use metros como unidad de medida de distancia
mi_punto <- sf::st_transform(mi_punto, 4488)
```

```r
# Genera un 'buffer' alrededor de tu punto de interés

# Elige de qué longitud en metros quieres que sea el radio de tu buffer (el área a
# generar alrededor de tu punto de interés)
mi_radio <- 2500


# Genera el buffer
mi_area <- sf::st_buffer(mi_punto,
                         dist = mi_radio,
                         nQuadSegs = 5)


# NOTA: Puedes controlar la forma del buffer con el argumento 'nQuadSegs'. nQuadSegs = 1
# produce un área cuadrada cuyos vértices se encuentran a la distancia del punto que
# estableciste en 'mi_radio'. nQuadSegs = 30 produce un área circular con radio en
# metros igual a 'mi_radio'. Considera que mientras más vértices tenga la forma de tu
# área, mayor es el riesgo de que la función con la que se descargan datos de GBIF
# tenga problemas. nQuadSegs = 5 produce un área cuasi-circular.


# OPCIONAL: Visualiza el área que acabas de generar alrededor de tu punto de interés
par(mfrow = c (1,1))
plot(mi_area)
plot(mi_punto, add = T)


# OPCIONAL: Calcula cuántas hectáreas tiene el área que acabas de generar
units::drop_units(sf::st_area(mi_area) / 10000)


# Re-proyecta a WGS 84
mi_area <- sf::st_transform(mi_area, crs = 4326)


# Obtén coordenadas en formato WKT
coords_wkt <- sf::st_as_text(mi_area)


# Genera mi_area() que puedas visualizar con el código en la sección 'Visualiza los
# registros descargados en un mapa simple')
mi_area <- sf::st_as_sf(mi_area)
sf::st_geometry(mi_area) <- "geometry"
```

Caso 2: Con 2 pares de coordenadas

Si tienes las coordenadas de un polígono mínimo (e.g. las esquinas superior izquierda e inferior derecha del rectángulo que engloba tu área de interés, o *bounding box*) genera el polígono correspondiente. Las coordenadas en el ejemplo de abajo aproximan el área de estudio de Adame, Moranchel, and Piedragil (2019).

```r
# Genera coordenadas WKT del polígono mínimo (bounding box) de tu área de interés


# IMPORTANTE: Cada par de coordenadas empieza con la longitud, después la latitud
mi_area <- c(-99.22639, 18.68389, -99.20972, 18.65611)


# Genera el polígono
Poly_Coord_df = data.frame(lon = mi_area[c(1, 3)],
                           lat = mi_area[c(2, 4)])


pol <- sf::st_polygon(list(cbind(Poly_Coord_df$lon[c(1,2,2,1,1)],
                                 Poly_Coord_df$lat[c(1,1,2,2,1)])))


# Asígnale proyección geográfica WGS 84
polc = sf::st_sfc(pol, crs = 4326)


# Obtén coordenadas en formato WKT
coords_wkt <- sf::st_as_text(polc)


# Genera mi_area() que puedas visualizar con el código en la sección 'Visualiza los
# registros descargados en un mapa simple')
mi_area <- sf::st_as_sf(polc)
sf::st_geometry(mi_area) <- "geometry"
```

Caso 3: Con archivo vectorial

Si tienes un Shapefile, Geopackage u otro archivo similar, puedes obtener sus coordenadas en WKT de la siguiente manera:

```r
# Genera coordenadas WKT a partir de un archivo vectorial

# Primero lee tu archivo especificando la dirección completa de la carpeta donde se
# encuentra.
# NOTA: No uses diagonales inversas o backslash (\) sino diagonales regulares (/).
mi_area <- sf::st_read("mi/directorio/mi_poligono.shp")


# OPCIONAL: Visualiza tu polígono.
# La primera línea dividirá el área de visualización para que después grafiques a su lado
# una versión simplificada del polígono
par(mfrow = c(1, 2))
plot(mi_area$geometry, main = "Polígono original")

# NOTA: Si esta línea te produce un error, es posible que debas reemplazar
#       'mi_area$geometry' por 'mi_area$geom'


# Asegúrate de que tu polígono tenga la proyección correcta (EPSG 4326)
```

```r
mi_area <- sf::st_transform(mi_area, 4326)


# Elimina la dimensión Z, si es que la tuviera
mi_area <- sf::st_zm(mi_area,
                     drop = TRUE,
                     what = "ZM")


# Si tu área tiene forma irregular entonces puede tener muchas coordenadas y eso le
# puede causar problemas a la función rgbif::occ_download() con la que se descargan datos
# de GBIF. Una solución es simplificar el polígono de manera que se reduzcan su
# número de vértices y coordenadas, pero cuidando que preserve su forma

mi_area_simple <- sf::st_simplify(mi_area,
                                  preserveTopology = T,
                                  dTolerance = 200)

# OPCIONAL: Grafica el polígono simplificado para compararlo visualmente
# con el original
plot(mi_area_simple$geometry, main = "Polígono simplificado")

# NOTA: Si esta línea te produce un error, es posible que debas reemplazar
#       'mi_area$geometry' por 'mi_area$geom'


# Si se perdió demasiado detalle, ejecuta nuevamente las líneas de arriba modificando
# primero el argumento 'dTolerance'. Mientras menor sea ese valor, mayor será el
# parecido con la figura original; por ejemplo, dTolerance = 0 no modifica la forma del
# área, mientras que dTolerance = 1000 la transforma por completo.


# Toma las coordenadas del polígono simplificado
mi_area_simple <- sf::st_sfc(mi_area_simple$geometry)


# Obtén las coordenadas en formato WKT
coords_wkt <- sf::st_as_text(mi_area_simple)
```

Descarga datos sin haberte registrado en GBIF (NO RECOMENDADO)

El paquete rgbif() ofrece una función que permite descargar datos de GBIF sin necesidad de haberse registrado previamente en la plataforma. Sin embargo, esta opción no es recomendada por diferentes razones. Una de ellas es que sólo se descargan una parte de los registros disponibles para tu área de estudio. Tú puedes controlar un poco la cantidad de registros a descargar usando el argumento 'mi_limite', pero en nuestra experiencia la única manera de saber si ya se descargaron todos los registros es a través del ensayo y error, verificando cada vez el número de registros. Además, la función tiene un límite máximo de registros que se pueden descargar, por lo que sería posible que este método no permita descargar todos los datos de grupos biológicos con muchos registros. Por el contrario, el método que usa tus datos de usuario de GBIF descarga todos los registros de tu grupo biológico de interés.

Además, si los datos que descargues van a ser utilizados en una publicación (artículo, tesis o reporte), es mejor descargar datos usando tu usuario y contraseña para poder citar la información y dar los créditos necesarios.

Para más información, consulta https://docs.ropensci.org/rgbif/articles/rgbif.html.

```
# Número máximo de registros que se van a descargar
mi_limite <- 100


# NOTA: La función occ_search tiene muchos más argumentos que los mostrados en este
#       ejemplo. Para explorarlos consulta ?rgbif::occ_search
mis_datos <- rgbif::occ_search(taxonKey = mi_grupo,
                               geometry = coords_wkt,
                               limit = mi_limite)

# extrae datos
mis_datos <-mis_datos$data
```

Paquetería utilizada

Paquete	Versión	Cita
base	4.3.2	R Core Team (2023)
bdc	1.1.4	B. Ribeiro et al. (2023)
devtools	2.4.5	Wickham et al. (2022)
doParallel	1.0.17	Corporation and Weston (2022)
foreach	1.5.2	Microsoft and Weston (2022)
iterators	1.0.14	Analytics and Weston (2022)
leaflet	2.2.1	Cheng et al. (2023)
mapview	2.11.2	Appelhans et al. (2023)
rgbif	3.7.8	Chamberlain and Boettiger (2017); Chamberlain et al. (2024)
rnaturalearthdata	0.1.0	South (2017)
rnaturalearthhires	0.2.1	South, Michael, and Massicotte (2023)
sf	1.0.14	Pebesma (2018); Pebesma and Bivand (2023)
spocc	1.2.2	Owens, Barve, and Chamberlain (2023)
tidyverse	2.0.0	Wickham et al. (2019)
units	0.8.2	Pebesma, Mailund, and Hiebert (2016)
usethis	2.2.2	Wickham et al. (2023)

Referencias

Adame, David R., Aarón Legaspi Moranchel, and Cesar D. Jiménez Piedragil. 2019. "Avifauna Del Parque Estatal Cerro de La Tortuga, Morelos, México." *Mesoamericana* 23 (1): 1–16. https://revistas.up.ac.pa/index.php/mesoamericana/article/view/774.

Analytics, Revolution, and Steve Weston. 2022. *iterators: Provides Iterator Construct.* https://CRAN.R-project.org/package=iterators.

Appelhans, Tim, Florian Detsch, Christoph Reudenbach, and Stefan Woellauer. 2023. *mapview: Interactive Viewing of Spatial Data in r.* https://CRAN.R-project.org/package=mapview.

Chamberlain, Scott, Vijay Barve, Dan Mcglinn, Damiano Oldoni, Peter Desmet, Laurens Geffert, and Karthik Ram. 2024. *rgbif: Interface to the Global Biodiversity Information Facility API.* https://CRAN.R-project.org/package=rgbif.

Chamberlain, Scott, and Carl Boettiger. 2017. “R Python, and Ruby Clients for GBIF Species Occurrence Data.” *PeerJ PrePrints*. https://doi.org/10.7287/peerj.preprints.3304v1.

Cheng, Joe, Barret Schloerke, Bhaskar Karambelkar, and Yihui Xie. 2023. *leaflet: Create Interactive Web Maps with the JavaScript “Leaflet” Library*. https://CRAN.R-project.org/package=leaflet.

Corporation, Microsoft, and Steve Weston. 2022. *doParallel: Foreach Parallel Adaptor for the “parallel” Package*. https://CRAN.R-project.org/package=doParallel.

Martínez-Morales, Miguel Ángel, Verónica Mendiola Islas, Iriana Zuria, Martha Cecilia Chávez Peón Hoffmann-Pinther, and Roberto Gabriel Campuzano Velasco. 2013. “La Conservación de Las Aves Más Allá de Las Áreas Naturales Protegidas: El Caso de La Avifauna Del Rancho Santa Elena, Hidalgo.” *Huitzil* 14 (2): 87–100. http://www.scielo.org.mx/scielo.php?script=sci_abstract&pid=S1870-74592013000200006&lng=es&nrm=iso&tlng=es.

Microsoft, and Steve Weston. 2022. *foreach: Provides Foreach Looping Construct*. https://CRAN.R-project.org/package=foreach.

Owens, Hannah, Vijay Barve, and Scott Chamberlain. 2023. *spocc: Interface to Species Occurrence Data Sources*. https://CRAN.R-project.org/package=spocc.

Pebesma, Edzer. 2018. “Simple Features for R: Standardized Support for Spatial Vector Data.” *The R Journal* 10 (1): 439–46. https://doi.org/10.32614/RJ-2018-009.

Pebesma, Edzer, and Roger Bivand. 2023. *Spatial Data Science: With applications in R*. Chapman and Hall/CRC. https://doi.org/10.1201/9780429459016.

Pebesma, Edzer, Thomas Mailund, and James Hiebert. 2016. “Measurement Units in R.” *R Journal* 8 (2): 486–94. https://doi.org/10.32614/RJ-2016-061.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ribeiro, Bruno R., Santiago José Elías Velazco, Karlo Guidoni-Martins, Geiziane Tessarolo, Lucas Jardim, Steven P. Bachman, and Rafael Loyola. 2022. “Bdc: A Toolkit for Standardizing, Integrating and Cleaning Biodiversity Data.” *Methods in Ecology and Evolution* 13 (7): 1421–28. https://doi.org/10.1111/2041-210X.13868.

Ribeiro, Bruno, Santiago Velazco, Karlo Guidoni-Martins, Geiziane Tessarolo, and Lucas Jardim. 2023. *bdc: Biodiversity Data Cleaning*. https://CRAN.R-project.org/package=bdc.

South, Andy. 2017. *rnaturalearthdata: World Vector Map Data from Natural Earth Used in “rnaturalearth”*. https://CRAN.R-project.org/package=rnaturalearthdata.

South, Andy, Schramm Michael, and Philippe Massicotte. 2023. *rnaturalearthhires: High Resolution World Vector Map Data from Natural Earth Used in Rnaturalearth*. https://docs.ropensci.org/rnaturalearthhires.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. “Welcome to the tidyverse.” *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Jennifer Bryan, Malcolm Barrett, and Andy Teucher. 2023. *usethis: Automate Package and Project Setup*. https://CRAN.R-project.org/package=usethis.

Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022. *devtools: Tools to Make Developing r Packages Easier*. https://CRAN.R-project.org/package=devtools.