One of the first things I took on when I started at the Cornell Lab of Ornithology was creating the auk R package for accessing eBird data. The entire eBird dataset can be downloaded as a massive text file, called the eBird Basic Dataset (EBD), and auk pulls out manageable chunks of the dataset based on various spatial, temporal, or taxonomic filters. I’m often asked “how do I extract data from within a polygon?” (usually “polygon” is replaced by “shapefile”, but I try to avoid that word since there’s good reasons to stop using shapefiles). Rather than answer these questions individually, I thought I’d do a quick post about how to do this with auk. Note that, at the time of posting, this requires auk version 0.4.1 or higher, which can be installed with:

install.packages("auk")

For more details on auk and eBird data in general, including how to get access to the EBD, it’s worth reading the first two chapters of the eBird Best Practices book. For the sake of speed and smaller file size, I’ll be working on a subset of the EBD containing all Northern Bobwhite records from 2019, which I obtained using the EBD custom download form,. However, everything I’ll show in this post works equally as well (just a lot slower!) on the full EBD. For this example, let’s say we want to extract all records from within a polygon defining Bird Conservation Region 27 (Southeastern Coastal Plains). A GeoPackage of this region is available on the GitHub repository for the eBird Best Practices book, place it in the data/ subdirectory of your RStudio project, then load it into R with:

library(sf)

library(auk)

library(dplyr)

poly <- read\_sf("data/gis-data.gpkg", layer = "bcr")

If you have a shapefile, replace "data/gis-data.gpkg" with the path to your shapefile and omit layer = "bcr". Now that we have a polygon, extracting eBird data is a two step process:

1. Extract data from the EBD that’s within a bounding box containing the polygons using the function auk\_bbox(). This is necessary because due to the way auk works under the hood, it can only filter to ranges of latitudes and longitudes.
2. Import the resulting data into R and further subset it to just the observations that fall within the polygon.

Fortunately, step 1 is made easier by auk\_bbox() accepting spatial sf or raster objects and automatically calculating the bounding box for you. For example,

auk\_ebd("data/ebd\_norbob\_201901\_201912\_relFeb-2020.txt") %>%

auk\_bbox(poly)

Input

EBD: /home/maelle/Documents/ropensci/roweb2/content/technotes/2020-04-16-extracting-ebird-data-from-a-polygon/data/ebd\_norbob\_201901\_201912\_relFeb-2020.txt

Output

Filters not executed

Filters

Species: all

Countries: all

States: all

BCRs: all

Bounding box: Lon -91.6 - -75.5; Lat 29.3 - 37.3

Date: all

Start time: all

Last edited date: all

Protocol: all

Project code: all

Duration: all

Distance travelled: all

Records with breeding codes only: no

Complete checklists only: no

Notice that the output of the above command says Bounding box: Lon -91.6 - -75.5; Lat 29.3 - 37.3, which are the bounds of the smallest square that contains the polygon. We’ll get all observations on complete checklists from May to August inside the bounding box of the polygon:

f\_out <- "data/ebd\_norbob\_poly.txt"

auk\_ebd("data/ebd\_norbob\_201901\_201912\_relFeb-2020.txt") %>%

# define filters

auk\_bbox(poly) %>%

auk\_date(c("\*-05-01", "\*-08-31")) %>%

auk\_complete() %>%

# compile and run filters

auk\_filter(f\_out)

The results were output to a file, which you can read in with read\_ebd().

ebd <- read\_ebd("data/ebd\_norbob\_poly.txt")

The data are now in a data frame and it’s time to proceed to step 2: further subset the data to only keep points within the polygon. First we’ll convert this data frame to a spatial sf object using the latitude and longitude columns, then well use st\_within() to identify the points within the polygon, and use this to subset the data frame. Note that we have to be careful with our coordinate reference system here: crs = 4326 specifies that the EBD data are in unprojected, lat-long coordinates and we use st\_transform() to ensure the polygons and points are in the coordinate reference system.

# convert to sf object

ebd\_sf <- ebd %>%

select(longitude, latitude) %>%

st\_as\_sf( coords = c("longitude", "latitude"), crs = 4326)

# put polygons in same crs

poly\_ll <- st\_transform(poly, crs = st\_crs(ebd\_sf))

# identify points in polygon

in\_poly <- st\_within(ebd\_sf, poly\_ll, sparse = FALSE)

although coordinates are longitude/latitude, st\_within assumes that they are planar

# subset data frame

ebd\_in\_poly <- ebd[in\_poly[, 1], ]

Finally, let’s create a simple map showing the EBD observations before (in black) and after (in green) subsetting the data to be within the polygon.

par(mar = c(0, 0, 0, 0))

plot(poly %>% st\_geometry(), col = "grey40", border = NA)

plot(ebd\_sf, col = "black", pch = 19, cex = 0.5, add = TRUE)

plot(ebd\_sf[in\_poly[, 1], ],

col = "forestgreen", pch = 19, cex = 0.5,

add = TRUE)

legend("top",

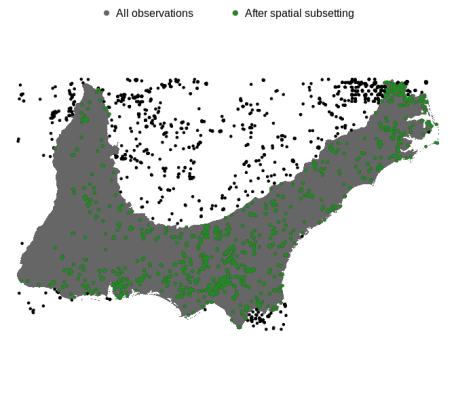
legend = c("All observations", "After spatial subsetting"),

col = c("grey40", "forestgreen"),

pch = 19,

bty = "n",

ncol = 2)



Looks like it worked! We got just the points within the polygon as intended. Two final notes:

1. If you’re working with the full EBD (a 200+ GB file), you’ll need to follow step 1 and subset the data using auk prior to importing into R. However, if you’ve used the custom download form to get an EBD subset, your file is likely small enough that you can read the data directly into R with read\_ebd() and skip straight to step 2.
2. If your intention is to eventually zero-fill the EBD to produce presence-absence data you’ll need to include the sampling event data file in the auk\_ebd(), subset both the EBD and sampling event data files separately to points within the polygon, the combine them together and zero-fill with auk\_zerofill().