Retrieving Köppen-Geiger Climate Classifications Based on GBIF Native Range Occurrence Data for Numerous Taxa

Kelsey Brock

5/10/2020

Set Up R

```
require(knitr)

## Loading required package: knitr

#get the needed packages
if(!require("pacman")){
    install.packages("pacman")
    library(pacman)}

## Loading required package: pacman

p_load( "data.table", "dplyr", "rworldmap", "sf", "rgdal", "tidyverse", "ggspatial", "rgbif", "readr", "purrr",  "magrittr", "taxize",  "kgc", "maptools", "geonames", "ggmap", "wicket", "ggplot2", "rnaturalearthdata", "sp", "rgeos", "hddtools", "smoothr", "units", "rmapshaper", "spThin")
```

First Section: Getting the GBIF taxon keys from species list

```
### Read in a csv file of occurrence data
SppList <- read.csv("sampledata.csv", header=T, sep=',', stringsAsFactors=F)
```

This is a sample data set. I have already retrieved IPNI IDs for all taxa (see https://github.com/kcbrock/The-HAPI-Project/tree/master/Plant_Accepted_Names_IDs for instructions), and the HAPI_ID-taxon is my own personal way of tracking species and avoiding errors with taxonomic synonyms, etc.

Use the Global Names Resolver in "taxize" to correct any spelling or formatting errors

```
#reorder alphabetically
SppList <- SppList[order(SppList$scientificNamePOW),]
#run GNR
gnr.long <- SppList$scientificNamePOW %>% gnr_resolve(data_source_ids = 11, best_match_only = TRUE, with_canonical_ranks = TRUE)
```

Retrieve GBIF "taxonkeys" for species list

```
gbif_taxon_keys <-
gnr.long %>%
pull("matched_name2") %>%
taxize::get_gbifid(method="backbone", db = "gbif", ask = TRUE,  phylum = "Tracheophyta", rank = "Species") #%>% # match

#merge data
inputname_IDs <- as.data.frame(cbind(gnr.long$user_supplied_name, gnr.long$matched_name2, gbif_taxon_keys))
colnames(inputname_IDs) = c("scientificNamePOW", "matched_name2", "gbif_taxon_key")
HawaiiGBIF <- merge(SppList, inputname_IDs, by = "scientificNamePOW" )
```

Second Section: Get the Occurrence Data from GBIF using polygons that represent their native ranges.

This section is not entirely automated because it's important to make sure that the data that is being called from various databases makes sense and to make sure you haven't accidentally assigned the wrong gbif key (as evidenced by error or low occurrence records).

```
# = starting at the first species
index = 1

# code to manually go on to the next row
taxonkey = HawaiiGBIF$gbif_taxon_key[index]
ipni_id = HawaiiGBIF$IPNI_ID[index]
HAPI_id = HawaiiGBIF$HAPI_ID_taxon[index]
species_name = HawaiiGBIF$scientificNamePOW[index]
index = index+1
species_name

## [1] "Euploca procumbens"
```

Get native range

Unfortunately, occurrence data isn't usually attributed with whether the plant is occuring in it's native or non-native range. This code gets the names of countries, states, continents and other various geographic regions from Kew's Plants of the World, and ask Google Maps to give you coordinates for the bounding box around these areas.

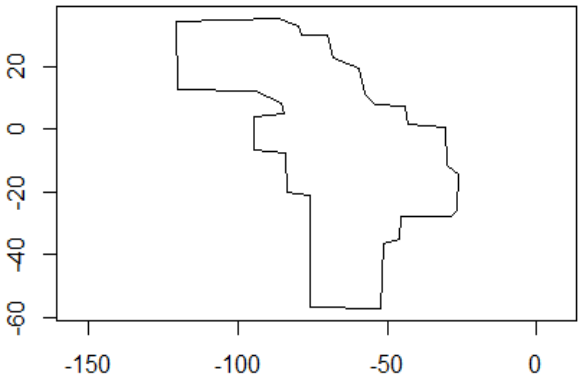
```
dist <- pow_lookup(id = ipni_id,include = "distribution")
dist <- as.data.frame(dist$meta$distribution$natives)

#fixing some google API errors
dist$name <- gsub("Jawa", "Java, Indonesia", dist$name)
dist$name <- gsub("Christmas I.", "Christmas Islands, Indonesia", dist$name)
dist$name <- gsub("Malaya", "Malaysia", dist$name)
dist$name <- gsub("China South-Central", "China", dist$name)
dist$name <- gsub("Argentina Northeast", "Argentina", dist$name)
dist$name <- gsub("Gilbert Is.", "Republic of Kiribati", dist$name)
dist$name <- gsub("Marianas", "Mariana Islands", dist$name)
dist$name <- gsub("Kazan-retto", "Ogasawara Islands", dist$name)
dist$name <- gsub("Ogasawara-shoto", "Ogasawara Islands", dist$name)
dist$name <- gsub("Nansei-shoto", "Nansei Islands", dist$name)
dist$name <- gsub("Victoria", "Victoria, Australia", dist$name)
dist$name <- gsub("Mexico Central", "Mexico", dist$name)
dist$name <- gsub("Mexico Northeast", "Mexico", dist$name)
dist$name <- gsub("Mexico Northwest", "Mexico", dist$name)
dist$name <- gsub("Netherlands Antilles", "Caribbean", dist$name)
dist$name <- gsub("Cape Provinces", "South Africa", dist$name)
dist$name <- gsub("Sumatera", "Sumatra", dist$name)
dist$name <- gsub("Is.", "Islands", dist$name)
dist$name <- gsub("Northwest European R", "Europe", dist$name)
dist$name <- gsub("Transcaucasus", "Caucasus Mountains", dist$name)
dist$name <- gsub("Manchuria", "Northeast Asia", dist$name)
dist$name <- gsub("Yugoslavia", "Serbia", dist$name)
dist$name <- gsub("Baltic States", "Lithuania", dist$name)
dist$name <- gsub("Christmas Islandsands, Indonesia", "Christmas Islands, Indonesia", dist$name)
dist$name <- gsub("Czechoslovakia", "Czech Republic", dist$name)
dist$name <- gsub("Togo", "Togo, Africa", dist$name)
dist$name <- gsub("Mariana Islandsands", "Marianas Islands", dist$name)
dist$name <- gsub("Nansei Islandsands", "Japan", dist$name)
register_google(key = "YOUR API KEY HERE")
df2 <- geocode(c(dist$name), output = "more")

df3 <- cbind(c(dist$name), df2)
df3 <- as.data.frame(df3)
head(df3)

##          c(dist$name)       lon        lat            type     loctype
## 1           Argentina -63.61667 -38.416097         country approximate
## 2 Argentina Northwest -63.61667 -38.416097         country approximate
## 3             Bahamas -77.39628  25.034280         country approximate
## 4              Belize -88.49765  17.189877         country approximate
## 5             Bolivia -63.58865 -16.290154         country approximate
## 6        Brazil North -58.37169  -2.095318 colloquial_area approximate
##                address      north     south      east      west
## 1            argentina -21.781046 -55.12502 -53.63748 -73.56036
## 2            argentina -21.781046 -55.12502 -53.63748 -73.56036
## 3          the bahamas  27.444104  21.42483 -70.67628 -81.25715
## 4               belize  18.495942  15.88562 -87.41270 -89.22759
## 5              bolivia  -9.669323 -22.89809 -57.45380 -69.64499
## 6 north region, brazil   5.271839 -13.69370 -45.69608 -73.99152
```

Check it out for weird locations – small bounding boxes of mistakes are okay. Are you missing large areas? If yes, add ","PLACENAME" to the geocode func.

Build a polygon of the native range

```
#some cleaning of na's and bad coordinates - this code just deletes them but you should check them out.
df3 <- df3[!is.na(df3$north),]
df3 <- subset(df3, df3$west < df3$east)

#make the bounding boxes
polybuild <- vector("list", nrow(df3))
  for(i in 1:nrow(df3)) {
  a <- as(raster::extent(df3$west[i], df3$east[i], df3$south[i], df3$north[i]), "SpatialPolygons")
  polybuild[[i]] <- a
}

# flatten multiple polygons into a single poly, and convert it into the wkt format used by GBIF
```

```
# also buffer the polys to get rid of holes (you cannot have nested polygons!)
wkt2 <- sp_convert(gSimplify(gBuffer(gUnionCascaded(list(polybuild, makeUniqueIDs = T) %>%
  flatten() %>%
  do.call(rbind, .)), byid = T, width = 2.5), tol = 1.5, topologyPreserve=FALSE), group=TRUE)

toplotit <- gSimplify(gBuffer(gUnionCascaded(list(polybuild, makeUniqueIDs = T) %>%
  flatten() %>%
  do.call(rbind, .)), byid = T, width =2.5), tol = 1.5, topologyPreserve=FALSE)
polygonarea <- raster::area(toplotit)

plot(toplotit, axes = TRUE)
```



Get the GBIF occurrences

```
temp <- occ_data(geometry = gsub("\n\\s+", "", wkt2), taxonKey=taxonkey, hasCoordinate=TRUE,
             hasGeospatialIssue=FALSE, basisOfRecord ='PRESERVED_SPECIMEN')


temp     <-subset(as.data.frame(temp$data))
temp <- distinct(temp, eventDate, decimalLatitude, decimalLongitude, .keep_all = TRUE )
temp['IPNI_ID']= ipni_id
temp['HAPI_ID']= HAPI_id
temp['NativeRange_Area']= polygonarea
temp     <-
  temp   %>% select(scientificName, locality, country,  basisOfRecord, eventDate, year, decimalLatitude, decimalLongitude
, basisOfRecord, issues, key, IPNI_ID, HAPI_ID, NativeRange_Area, coordinateUncertaintyInMeters)
nrow(temp)

## [1] 427
```

Second Section: Get the Occurrence Data from GBIF using polygons that represent their native ranges.

Thin Occurrence Data

It's no use getting the climate data for points that are right on top of each other. The below considers points within a 2-km radius of each other duplicates and removes them.

```
temp <- as.data.frame(transform(temp,latlon=paste0(decimalLatitude,decimalLongitude)))
temp <-distinct(temp, latlon, .keep_all = TRUE)
thinned <- thin(temp, verbose = FALSE,
                        lat.col = "decimalLatitude",
                        long.col = "decimalLongitude",
                        spec.col = "HAPI_ID",
                        thin.par = 2,
                        reps = 1,
                        write.files = FALSE,
                        write.log.file = FALSE,
                        locs.thinned.list.return = TRUE)

thinned <- as.data.frame(transform(thinned,latlon=paste0(Latitude,Longitude)))
temp <- merge(thinned, temp, by = "latlon", all.x = TRUE)
nrow(temp)

## [1] 364
```

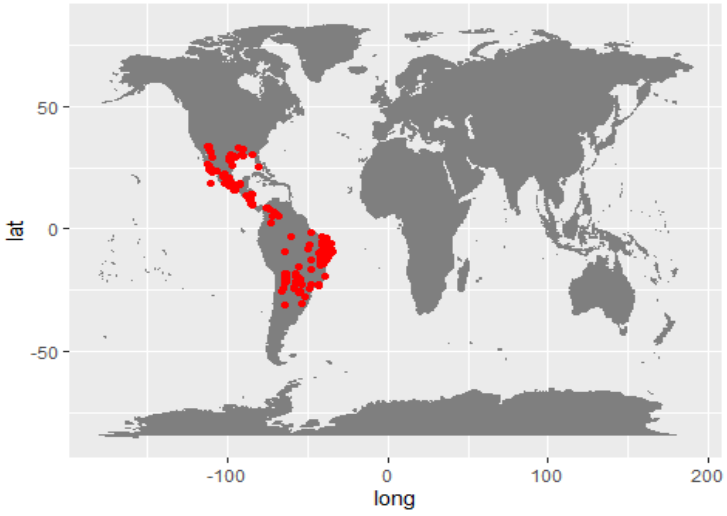
Third Section: Get the Köppen-Geiger Climate Classification

The below code pulls the climate zone classification for each occurrence, and then adds a column that shows the total proportion of each climate zone for each species.

```
  # Query Coarse Resolution
  temp2 <- data.frame(temp, rndCoord.lon = RoundCoordinates(temp$decimalLongitude),rndCoord.lat =
         RoundCoordinates(temp$decimalLatitude))
  temp2 <- data.frame(temp2,CZ=LookupCZ(temp2))
  colnames(temp2)[which(colnames(temp2)=='rndCoord.lon')] <- 'rndCoord.lon.course'
  colnames(temp2)[which(colnames(temp2)=='rndCoord.lat')] <- 'rndCoord.lat.course'
temp3 <- prop.table(table(temp2$HAPI_ID,temp2$CZ))
temp3 <- as.data.frame.matrix(temp3)
temp3 <- setDT(temp3, keep.rownames = TRUE)[]
temp3 <- merge(temp2, temp3, by.x = "HAPI_ID", by.y = "rn")
```

Mapping your occurrences

```
mapWorld <- borders("world", colour="gray50", fill="gray50")
messy_map <- ggplot()+
              mapWorld+
              geom_point(temp3,
              mapping = aes(x = decimalLongitude,
              y = decimalLatitude),
              col = "red")
messy_map
```



Save the Data

```
#species specific files
newname <- sub(" ", "_", species_name)
write.csv(temp3, file=paste(newname, ".csv", sep=""), row.names = F)
saveRDS(temp3, file=paste(newname, ".RDS", sep=""))

#the mega all-species file
mega_GBIF_CZ_df <- read.csv("mega_file.csv", header=T, sep=',', stringsAsFactors=F)
mega_GBIF_CZ_df <- dplyr::bind_rows(mutate_all(mega_GBIF_CZ_df, as.character), mutate_all(temp3, as.character))
write.csv(mega_GBIF_CZ_df, file = "mega_file.csv", row.names = FALSE)
saveRDS(mega_GBIF_CZ_df, file = "mega_file.RDS")
```

END!