Source code for *Using citizen science to parse climatic and landcover influences on bird occupancy within a tropical biodiversity hotspot*

Vijay Ramesh   Pratik R. Gupte   Morgan W. Tingley   VV Robin   Ruth DeFries

2021-06-17

Contents

1 Introduction

This is the readable version containing analysis that models associations between environmental predictors (climate and landcover) and citizen science observations of birds across the Nilgiri and Anamalai Hills of the Western Ghats Biodiversity Hotspot.

Methods and format are derived from Strimas-Mackey et al..

1.1 Attribution

Please contact the following in case of interest in the project.

- Vijay Ramesh (lead author)
  - PhD student, Columbia University
- Pratik Gupte (repo maintainer)
  - PhD student, University of Groningen

1.2 Data access

The data used in this work are available from eBird.

1.3 Data processing

The data processing for this project is described in the following sections. Navigate through them using the links in the sidebar.

1.4 Main Text Figure 1

Figure prepared in QGIS 3.10.

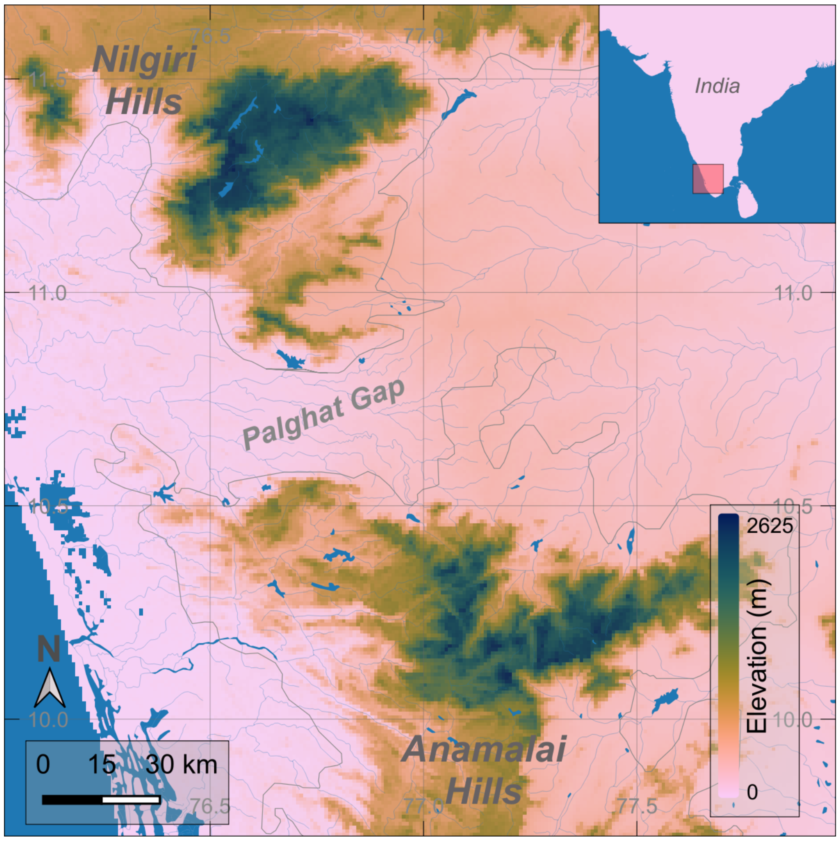

Figure 1: A shaded relief of the study area - the Nilgiri and the Anamalai hills are shown in this figure. This map was made using the SRTM digital elevation model at a spatial resolution of 1km and data from Natural Earth were used to outline boundaries of water bodies.

2 Preparing eBird Data

2.1 Prepare libraries and data sources

Here, we will load the necessary libraries required for preparing the eBird data. Please download the latest versions of the eBird Basic Dataset (for India) and the eBird Sampling dataset from https://ebird.org/data/download.

```
# load libraries
library(tidyverse)
library(readr)
library(sf)
library(auk)
library(readxl)
library(lubridate)

# custom sum function
sum.no.na <- function(x) {
  sum(x, na.rm = T)
}

# set file paths for auk functions
# To use these two datasets, please download the latest versions from https://ebird.org/data/download and set t

f_in_ebd <- file.path("data/ebd_IN_relMar-2021.txt")
f_in_sampling <- file.path("data/ebd_sampling_relMar-2021.txt")
```

2.2 Filter data

Insert the list of species that we will be analyzing in this study. We initially chose those species that occurred in at least 5% of all checklists across 50% of the 25 x 25 km cells from where they have been reported, resulting in a total of 93 species. To arrive at this final list of species, we carried out further pre-processing which can be found in Section 2 of the Supplementary material.

```
# add species of interest
specieslist <- read.csv("data/species_list.csv")
speciesOfInterest <- as.character(specieslist$scientific_name)
```

Here, we set broad spatial filters for the states of Kerala, Tamil Nadu and Karnataka and keep only those checklists for our list of species that were reported between 1st Jan 2013 and 31st Dec 2020.

```
# run filters using auk packages
ebd_filters <- auk_ebd(f_in_ebd, f_in_sampling) %>%
  auk_species(speciesOfInterest) %>%
  auk_country(country = "IN") %>%
  auk_state(c("IN-KL", "IN-TN", "IN-KA")) %>%
  # Restricting geography to TamilNadu, Kerala & Karnataka
  auk_date(c("2013-01-01", "2020-12-31")) %>%
  auk_complete()

# check filters
ebd_filters
```

Below code need not be run if it has been filtered once already and the above path leads to the right dataset. NB: This is a computation heavy process, run with caution.

```
# specify output location and perform filter
f_out_ebd <- "data/01_ebird-filtered-EBD-westernGhats.txt"
```

```r
f_out_sampling <- "data/01_ebird-filtered-sampling-westernGhats.txt"

ebd_filtered <- auk_filter(ebd_filters,
  file = f_out_ebd,
  file_sampling = f_out_sampling, overwrite = TRUE
)
```

2.3 Process filtered data

The data has been filtered above using the auk functions. We will now work with the filtered checklist observations (Please note that we have not yet spatially filtered the checklists to the confines of our study area, which is the Nilgiris and the Anamalai hills. This step is carried out further on).

```r
# read in the data
ebd <- read_ebd(f_out_ebd)
```

eBird checklists only suggest whether a species was reported at a particular location. To arrive at absence data, we use a process known as zero-filling (Johnston et al. 2019), wherein a new dataframe is created with a 0 marked for each checklist when the bird was not observed.

```r
# fill zeroes
zf <- auk_zerofill(f_out_ebd, f_out_sampling)
new_zf <- collapse_zerofill(zf)
```

Let us now choose specific columns necessary for further analysis.

```r
# choose columns of interest
columnsOfInterest <- c(
  "checklist_id", "scientific_name", "common_name",
  "observation_count", "locality", "locality_id",
  "locality_type", "latitude", "longitude",
  "observation_date", "time_observations_started",
  "observer_id", "sampling_event_identifier",
  "protocol_type", "duration_minutes",
  "effort_distance_km", "effort_area_ha",
  "number_observers", "species_observed",
  "reviewed"
)

# make list of presence and absence data and choose cols of interest
data <- list(ebd, new_zf) %>%
  map(function(x) {
    x %>% select(one_of(columnsOfInterest))
  })

# remove zerofills to save working memory
rm(zf, new_zf)
gc()

# check for presences and absence in absences df, remove essentially the presences df which may lead to erroneo
data[[2]] <- data[[2]] %>% filter(species_observed == F)
```

2.4 Spatial filter

A spatial filter is now supplied to further restrict our list of observations to the confines of the Nilgiris and the Anamalai hills of the Western Ghats biodiversity hotspot.

```
# load shapefile of the study area
library(sf)
hills <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp")

# write a prelim filter by bounding box
box <- st_bbox(hills)

# get data spatial coordinates
dataLocs <- data %>%
  map(function(x) {
    select(x, longitude, latitude) %>%
      filter(between(longitude, box["xmin"], box["xmax"]) &
        between(latitude, box["ymin"], box["ymax"]))
  }) %>%
  bind_rows() %>%
  distinct() %>%
  st_as_sf(coords = c("longitude", "latitude")) %>%
  st_set_crs(4326) %>%
  st_intersection(hills)

# get simplified data and drop geometry
dataLocs <- mutate(dataLocs, spatialKeep = T) %>%
  bind_cols(., as_tibble(st_coordinates(dataLocs))) %>%
  st_drop_geometry()

# bind to data and then filter
data <- data %>%
  map(function(x) {
    left_join(x, dataLocs, by = c("longitude" = "X", "latitude" = "Y")) %>%
      filter(spatialKeep == T) %>%
      select(-Id, -spatialKeep)
  })
```

Save temporary data created so far.

```
# save a temp data file
save(data, file = "data/01_data_temp.rdata")
```

2.5 Handle presence data

Further pre-processing is required in the case of many checklists where species abundance is often unknown and an 'X' is denoted in such cases. Here, we convert all 'X' notations to a 1, suggesting a presence (as we are not concerned with abundance data in this analysis). We also removed those checklists where the duration in minutes is either not recorded or listed as zero. Lastly, we added an sampling effort based filter following (Johnston et al. 2019), wherein we considered only those checklists with duration in minutes is less than 300 and distance in kilometres traveled is less than 5km. Lastly, we excluded those group checklists where the number of observers was greater than 10. Upon receiving comments from reviewers, we restrict all our checklists between December 1st and May 31st (non-rainy months)and checklists recorded between 5am and 7pm.

```
# in the first set, replace X, for presences, with 1
data[[1]] <- data[[1]] %>%
  mutate(observation_count = ifelse(observation_count == "X",
    "1", observation_count
  ))

# remove records where duration is 0
```

```r
data <- map(data, function(x) filter(x, duration_minutes > 0))

# group data by site and sampling event identifier
# then, summarise relevant variables as the sum
dataGrouped <- map(data, function(x) {
  x %>%
    group_by(sampling_event_identifier) %>%
    summarise_at(
      vars(
        duration_minutes, effort_distance_km,
        effort_area_ha
      ),
      list(sum.no.na)
    )
})

# bind rows combining data frames, and filter
dataGrouped <- bind_rows(dataGrouped) %>%
  filter(
    duration_minutes <= 300,
    effort_distance_km <= 5,
    effort_area_ha <= 500
  )

# get data identifiers, such as sampling identifier etc
dataConstants <- data %>%
  bind_rows() %>%
  select(
    sampling_event_identifier, time_observations_started,
    locality, locality_type, locality_id,
    observer_id, observation_date, scientific_name,
    observation_count, protocol_type, number_observers,
    longitude, latitude
  )

# join the summarised data with the identifiers,
# using sampling_event_identifier as the key
dataGrouped <- left_join(dataGrouped, dataConstants,
  by = "sampling_event_identifier"
)

# remove checklists or seis with more than 10 obervers
count(dataGrouped, number_observers > 10) # count how many have 10+ obs
dataGrouped <- filter(dataGrouped, number_observers <= 10)

# keep only checklists between 5AM and 7PM
dataGrouped <- filter(dataGrouped, time_observations_started >= "05:00:00" & time_observations_started <= "19:0

# keep only checklists between December 1st and May 31st
dataGrouped <- filter(dataGrouped, month(observation_date) %in% c(1, 2, 3, 4, 5, 12))
```

2.6 Add decimal time

We added a column where time is denoted in decimal hours since midnight.

```r
# assign present or not, and get time in decimal hours since midnight
library(lubridate)
time_to_decimal <- function(x) {
  x <- hms(x, quiet = TRUE)
  hour(x) + minute(x) / 60 + second(x) / 3600
}

# will cause issues if using time obs started as a linear effect and not quadratic
dataGrouped <- mutate(dataGrouped,
  pres_abs = observation_count >= 1,
  decimalTime = time_to_decimal(time_observations_started)
)

# check class of dataGrouped, make sure not sf
assertthat::assert_that(!"sf" %in% class(dataGrouped))
```

The above data is saved to a file.

```r
# save a temp data file
save(dataGrouped, file = "data/01_data_prelim_processing.Rdata")
```

3 Preparing Environmental Predictors

In this script, we processed climatic and landscape predictors for occupancy modeling.

All climatic data was obtained from https://chelsa-climate.org/bioclim/ All landscape data was derived from a high resolution land cover map (Roy et al 2015). This map provides sufficient classes to achieve a high land cover resolution.

The goal here is to resample all rasters so that they have the same resolution of 1km cells. We also tested for spatial autocorrelation among climatic predictors, which can be found in Section 4 of the Supplementary Material.

3.1 Prepare libraries

We load some common libraries for raster processing and define a custom mode function.

```r
# load libs
library(raster)
library(stringi)
library(glue)
library(gdalUtils)
library(purrr)
library(dplyr)
library(tidyr)
library(tibble)

# for plotting
library(viridis)
library(tmap)
library(scales)
library(ggplot2)
library(patchwork)

# prep mode function to aggregate
funcMode <- function(x, na.rm = T) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

# a basic test
assertthat::assert_that(funcMode(c(2, 2, 2, 2, 3, 3, 3, 4)) == as.character(2),
  msg = "problem in the mode function"
) # works

# get ci func
ci <- function(x) {
  qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
}
```

3.2 Prepare spatial extent

We prepare a 30km buffer around the boundary of the study area. This buffer will be used to mask the landscape rasters.The buffer procedure is done on data transformed to the UTM 43N CRS to avoid distortions.

```r
# load hills
library(sf)
hills <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp")
hills <- st_transform(hills, 32643)
buffer <- st_buffer(hills, 3e4) %>%
  st_transform(4326)
```

3.3 Prepare terrain rasters

We prepare the elevation data which is an SRTM raster layer, and derive the slope and aspect from it after cropping it to the extent of the study site buffer. Please download the latest version of the SRTM raster layer from https://www.worldclim.org/data/worldclim21.html

```r
# load elevation and crop to hills size, then mask by hills
alt <- raster("data/spatial/Elevation/alt") # this layer is not added to github as a result of its large size a
alt.hills <- crop(alt, as(buffer, "Spatial"))
rm(alt)
gc()

# get slope and aspect
slopeData <- terrain(x = alt.hills, opt = c("slope", "aspect"))
elevData <- raster::stack(alt.hills, slopeData)
rm(alt.hills)
gc()
```

3.4 Prepare Bioclim 4a and Bioclim 15

```r
# list chelsa files
chelsaFiles <- list.files("data/chelsa/", recursive = TRUE, pattern = "tif")

# load Bio 1 and 12
bio_1 <- raster("data/chelsa/envicloud/chelsa/chelsa_V1/climatologies/bio/CHELSA_bio10_01.tif")
bio_12 <- raster("data/chelsa/envicloud/chelsa/chelsa_V1/climatologies/bio/CHELSA_bio10_12.tif")
```

3.5 Prepare CHELSA rasters

CHELSA rasters can be downloaded using the `get_chelsa.sh` shell script, which is a `wget` command pointing to the `envidatS3.txt` file.

3.5.1 Prepare BIOCLIM 4a and 15

We prepare the CHELSA rasters for seasonality in temperature (Bio 4a) and seasonality in precipitation (Bio 15) in the same way, reading them in, cropping them to the study site buffer extent, and handling the temperature layer values which we divide by 10. The CHELSA rasters can be downloaded from https://chelsa-climate.org/bioclim/

```r
# list chelsa files
# the chelsa data can be downloaded from the aforementioned link. They haven't been uploaded to github as a res
chelsaFiles <- list.files("data/chelsa/",
  full.names = TRUE,
  recursive = TRUE,
  pattern = "bio10"
)

# gather chelsa rasters
chelsaData <- purrr::map(chelsaFiles, function(chr) {
  a <- raster(chr)
  crs(a) <- crs(elevData)
  a <- crop(a, as(buffer, "Spatial"))
  return(a)
})

# divide temperature by 10
chelsaData[[1]] <- chelsaData[[1]] / 10

# stack chelsa data
chelsaData <- raster::stack(chelsaData)
```

3.5.2 Prepare BIOCLIM 4a

```r
if (file.exists("data/chelsa/CHELSA_bio10_4a.tif")) {
  message("Bio 4a already exists, will be overwritten")
}
```

Bioclim 4a, the coefficient of variation temperature seasonality is calculated as

$$Bio\ 4a = \frac{SD\{Tkavg_1, ...Tkavg_{12}\}}{(Bio\ 1 + 273.15)} \times 100$$

where $Tkavg_i = (Tkmin_i + Tkmax_i)/2$

Here, we use only the months of December and Jan – May for winter temperature variation.

```r
# list rasters by pattern
patterns <- c("tmin", "tmax")

# list the filepaths
tkAvg <- map(patterns, function(pattern) {
  # list the paths
  files <- list.files(
    path = "data/chelsa",
    full.names = TRUE,
    recursive = TRUE,
    pattern = pattern
  )
})
```

```r
# print crs elev data for sanity check --- basic WGS84
crs(elevData)

# now run over the paths and read as rasters and crop by buffer
tkAvg <- map(tkAvg, function(paths) {
  # going over the file paths, read them in as rasters, convert CRS and crop
  tempData <- map(paths, function(path) {
    # read in
    a <- raster(path)
    # assign crs
    crs(a) <- crs(elevData)
    # crop by buffer, will throw error if CRS doesn't match
    a <- crop(a, as(buffer, "Spatial"))
    # return a
    a
  })
  # convert each to kelvin, first dividing by 10 to get celsius
  tempData <- map(tempData, function(tmpRaster) {
    tmpRaster <- (tmpRaster / 10) + 273.15
  })
})

# assign names
names(tkAvg) <- patterns

# go over the tmin and tmax and get the average monthly temp
tkAvg <- map2(tkAvg[["tmin"]], tkAvg[["tmax"]], function(tmin, tmax) {
  # return the mean of the corresponding tmin and tmax
  # still in kelvin
  calc(stack(tmin, tmax), fun = mean)
})

# calculate Bio 4a
bio_4a <- (calc(stack(tkAvg), fun = sd) / (chelsaData[[1]] + 273.15)) * 100
names(bio_4a) <- "CHELSA_bio10_4a"
# save bio_4a
writeRaster(bio_4a, filename = "data/chelsa/CHELSA_bio10_4a.tif", overwrite = T)
```

160 ### 3.5.3 Prepare Bioclim 15

```r
if (file.exists("data/chelsa/CHELSA_bio10_15.tif")) {
  message("Bio 15 already exists, will be overwritten")
}
```

161 Bioclim 15, the coefficient of variation precipitation (in our area, rainfall) seasonality is calculated as

$$Bio\ 15 = \frac{SD\{PPT_1, \ldots PPT_{12}\}}{1 + (Bio\ 12/12)} \times 100$$

162 where PPT_i is the monthly precipitation.

163 Here, we use only the months of December and Jan – May for winter rainfall variation.

```r
# list rasters by pattern
pattern <- "prec"
```

```r
# list the filepaths
pptTotal <- list.files(
  path = "data/chelsa",
  full.names = TRUE,
  recursive = TRUE,
  pattern = pattern
)

# print crs elev data for sanity check --- basic WGS84
crs(elevData)

# now run over the paths and read as rasters and crop by buffer
pptTotal <- map(pptTotal, function(path) {
  a <- raster(path)
  # assign crs
  crs(a) <- crs(elevData)
  # crop by buffer, will throw error if CRS doesn't match
  a <- crop(a, as(buffer, "Spatial"))
  # return a
  a
})

# calculate Bio 4a
bio_15 <- (calc(stack(pptTotal), fun = sd) / (1 + (chelsaData[[2]] / 12))) * 100
names(bio_15) <- "CHELSA_bio10_15"
# save bio_4a
writeRaster(bio_15, filename = "data/chelsa/CHELSA_bio10_15.tif", overwrite = T)
```

3.5.4 Stack terrain and climate

We stack the terrain and climatic rasters.

```r
# stack rasters for efficient reprojection later
env_data <- stack(elevData, bio_4a, bio_15)
```

3.6 Resample landcover from 10m to 1km

We read in a land cover classified image and resample that using the mode function to a 1km resolution. Please note that the resampling process need not be carried out as it has been done already and the resampled raster can be loaded with the subsequent code chunk.

```r
# read in landcover raster location
landcover <- "data/landUseClassification/landcover_roy_2015/"

# read in and crop
landcover <- raster(landcover)
buffer_utm <- st_transform(buffer, 32643)
landcover <- crop(
  landcover,
  as(
    buffer_utm,
    "Spatial"
  )
)
```

```r
# read reclassification matrix
reclassification_matrix <- read.csv("data/landUseClassification/LandCover_ReclassifyMatrix_2015.csv")
reclassification_matrix <- as.matrix(reclassification_matrix[, c("V1", "To")])

# reclassify
landcover_reclassified <- reclassify(
  x = landcover,
  rcl = reclassification_matrix
)

# write to file
writeRaster(landcover_reclassified,
  filename = "data/landUseClassification/landcover_roy_2015_reclassified.tif",
  overwrite = TRUE
)

# check reclassification
plot(landcover_reclassified)

# get extent
e <- bbox(raster(landcover))

# init resolution
res_init <- res(raster(landcover))

# res to transform to 1000m
res_final <- res_init * (1000 / res_init)

# use gdalutils gdalwarp for resampling transform
# to 1km from 10m
gdalUtils::gdalwarp(
  srcfile = "data/landUseClassification/landcover_roy_2015_reclassified.tif",
  dstfile = "data/landUseClassification/lc_01000m.tif",
  tr = c(res_final), r = "mode", te = c(e)
)
```

We compare the frequency of landcover classes between the original raster and the resampled 1km raster to be certain that the resampling has not resulted in drastic misrepresentation of the frequency of any landcover type. This comparison is made using the figure below.

3.7 Resample other rasters to 1km

We now resample all other rasters to a resolution of 1km.

3.7.1 Read in resampled landcover

Here, we read in the 1km landcover raster and set 0 to NA.

```r
lc_data <- raster("data/landUseClassification/lc_01000m.tif")
lc_data[lc_data == 0] <- NA
```

3.7.2 Reproject environmental data using landcover as a template

```r
# resample to the corresponding landcover data
env_data_resamp <- projectRaster(
  from = env_data, to = lc_data,
  crs = crs(lc_data), res = res(lc_data)
)

# export as raster stack
land_stack <- stack(env_data_resamp, lc_data)

# get names
land_names <- glue('data/spatial/landscape_resamp{c("01")}_km.tif')

# write to file
writeRaster(land_stack, filename = as.character(land_names), overwrite = TRUE)
```

3.8 Temperature and rainfall in relation to elevation

3.8.1 Load resampled environmental rasters

```r
# read landscape prepare for plotting
landscape <- stack("data/spatial/landscape_resamp01_km.tif")

# get proper names
elev_names <- c("elev", "slope", "aspect")
chelsa_names <- c("bio_4a", "bio_15")

names(landscape) <- glue('{c(elev_names, chelsa_names, "landcover")}')

# make duplicate stack
land_data <- landscape[[c("elev", chelsa_names)]]

# convert to list
land_data <- as.list(land_data)

# map get values over the stack
land_data <- purrr::map(land_data, getValues)
names(land_data) <- c("elev", chelsa_names)

# conver to dataframe and round to 100m
land_data <- bind_cols(land_data)
land_data <- drop_na(land_data) %>%
  mutate(elev_round = plyr::round_any(elev, 200)) %>%
  dplyr::select(-elev) %>%
  pivot_longer(
    cols = contains("bio"),
    names_to = "clim_var"
  ) %>%
  group_by(elev_round, clim_var) %>%
  summarise_all(.funs = list(~ mean(.), ~ ci(.)))
```

Figure code is hidden in versions rendered as HTML or PDF.

3.9 Land cover type in relation to elevation

```r
# get data from landscape rasters
lc_elev <- tibble(
  elev = getValues(landscape[["elev"]]),
  landcover = getValues(landscape[["landcover"]])
)

# process data for proportions
lc_elev <- lc_elev %>%
  filter(!is.na(landcover), !is.na(elev)) %>%
  mutate(elev = plyr::round_any(elev, 100)) %>%
  count(elev, landcover) %>%
  group_by(elev) %>%
  mutate(prop = n / sum(n))

# fill out lc elev
lc_elev_canon <- crossing(
  elev = unique(lc_elev$elev),
  landcover = unique(lc_elev$landcover)
)

# bind with lcelev
lc_elev <- full_join(lc_elev, lc_elev_canon)

# convert NA to zero
lc_elev <- replace_na(lc_elev, replace = list(n = 0, prop = 0))
```

Figure code is hidden in versions rendered as HTML and PDF.

3.10 Main Text Figure 2

4 Preparing Observer Expertise Scores

Differences in local avifaunal expertise among citizen scientists can lead to biased species detection when compared with data collected by a consistent set of trained observers (van Strien, van Swaay, and Termaat 2013). Including observer expertise as a detection covariate in occupancy models using eBird data can help account for this variation (Johnston et al. 2018). Observer-specific expertise in local avifauna was calculated following (Kelling et al. 2015) as the normalized predicted number of species reported by an observer after 60 minutes of sampling across the most common land cover type within the study area. This score was calculated by examining checklists from anonymized observers across the study area. We modified Kelling et al. (2015) formulation by including only observations of the 82 species of interest in our calculations. An observer with a higher number of species of interest reported within 60 minutes would have a higher observer-specific expertise score, with respect to the study area.

Plots with respect to how observer expertise varied over time (2013-2020) for the list of species considered in this study (across the study area alone) can be accessed in Section 7 of the Supplementary Material.

4.1 Prepare libraries

```
# load libs
library(data.table)
library(readxl)
library(magrittr)
library(stringr)
library(dplyr)
library(tidyr)
library(auk)

# get decimal time function
library(lubridate)
```

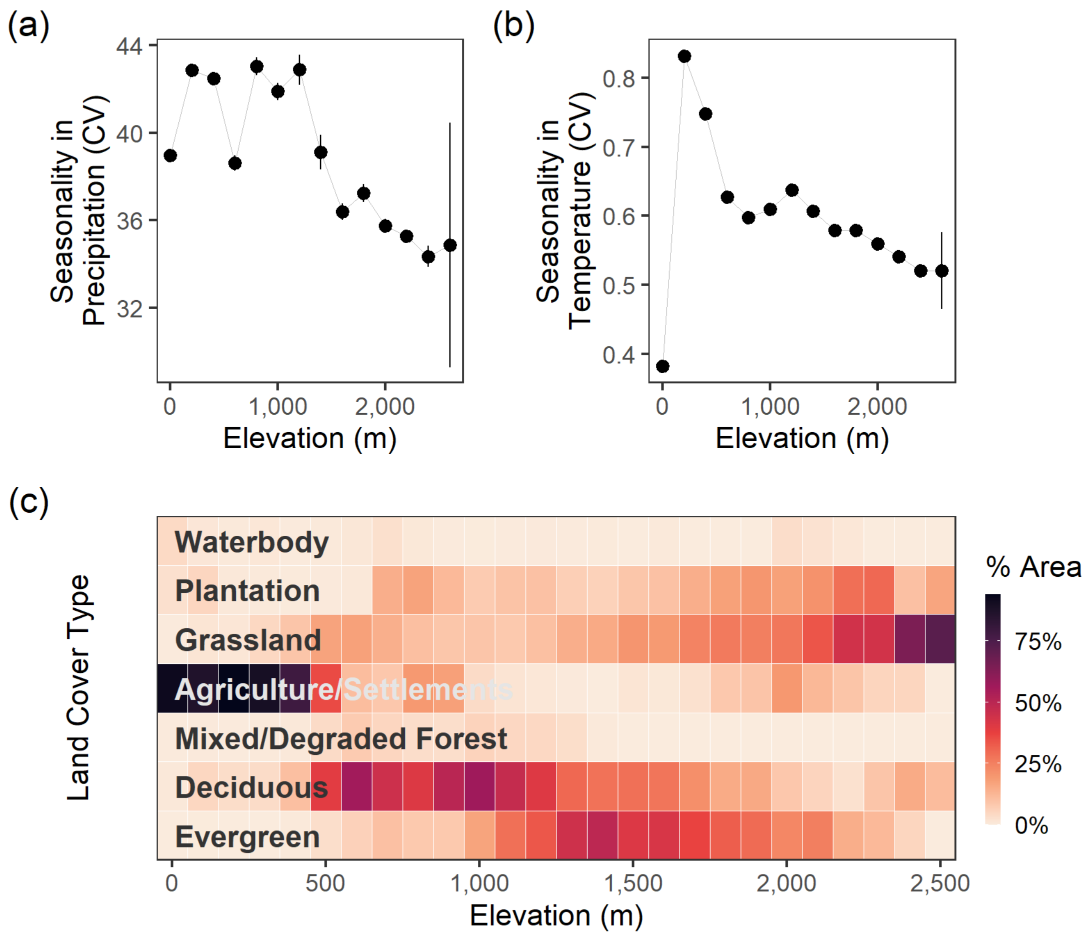

Figure 2: (a) Seasonality in precipitation and seasonality in temperature as a function of elevation is shown here. The coefficient of variation was calculated. (b) The proportion of land cover types varied across the study area as shown in this panel.

```
time_to_decimal <- function(x) {
  x <- lubridate::hms(x, quiet = TRUE)
  lubridate::hour(x) + lubridate::minute(x) / 60 + lubridate::second(x) / 3600
}
```

4.2 Prepare data

Here, we go through the data preparation process again because we might want to assess observer expertise over a larger area than the study site.

```
# Read in shapefile of study area to subset by bounding box
library(sf)
wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
  st_transform(32643)

# set file paths for auk functions
f_in_ebd <- file.path("data/01_ebird-filtered-EBD-westernGhats.txt")
f_in_sampling <- file.path("data/01_ebird-filtered-sampling-westernGhats.txt")

# run filters using auk packages
ebd_filters <- auk_ebd(f_in_ebd, f_in_sampling) %>%
  auk_country(country = "IN") %>%
  auk_state(c("IN-KL", "IN-TN", "IN-KA")) %>%
  # Restricting geography to TamilNadu, Kerala & Karnataka
  auk_date(c("2013-01-01", "2020-12-31")) %>%
  auk_complete()

# check filters
ebd_filters

# specify output location and perform filter
f_out_ebd <- "data/ebird_for_expertise.txt"
f_out_sampling <- "data/ebird_sampling_for_expertise.txt"

ebd_filtered <- auk_filter(ebd_filters,
  file = f_out_ebd,
  file_sampling = f_out_sampling, overwrite = TRUE
)
```

Load in the filtered data and columns of interest.

```
## Process filtered data
# read in the data
ebd <- fread(f_out_ebd)
names <- names(ebd) %>%
  stringr::str_to_lower() %>%
  stringr::str_replace_all(" ", "_")

setnames(ebd, names)
# choose columns of interest
columnsOfInterest <- c(
  "global_unique_identifier", "scientific_name", "observation_count",
  "locality", "locality_id", "locality_type", "latitude",
  "longitude", "observation_date",
  "time_observations_started", "observer_id",
  "sampling_event_identifier", "protocol_type",
```

```r
  "duration_minutes", "effort_distance_km", "effort_area_ha",
  "number_observers", "all_species_reported", "reviewed"
)

ebd <- setDF(ebd) %>%
  as_tibble() %>%
  dplyr::select(one_of(columnsOfInterest))

setDT(ebd)

# remove checklists or seis with more than 10 obervers
ebd <- filter(ebd, number_observers <= 10)

# keep only checklists between 5AM and 7PM
ebd <- filter(ebd, time_observations_started >= "05:00:00" & time_observations_started <= "19:00:00")

# keep only checklists between December 1st and May 31st
ebd <- filter(ebd, month(observation_date) %in% c(1, 2, 3, 4, 5, 12))
```

201 4.3 Spatially explicit filter on checklists

```r
# get checklist locations
ebd_locs <- ebd[, .(longitude, latitude)]
ebd_locs <- setDF(ebd_locs) %>% distinct()
ebd_locs <- st_as_sf(ebd_locs,
  coords = c("longitude", "latitude")
) %>%
  `st_crs<-`(4326) %>%
  bind_cols(as_tibble(st_coordinates(.))) %>%
  st_transform(32643) %>%
  mutate(id = 1:nrow(.))

# check whether to include
to_keep <- unlist(st_contains(wg, ebd_locs))

# filter locs
ebd_locs <- filter(ebd_locs, id %in% to_keep) %>%
  bind_cols(as_tibble(st_coordinates(st_as_sf(.)))) %>%
  st_drop_geometry()
names(ebd_locs) <- c("longitudeWGS", "latitudeWGS", "id", "longitudeUTM", "latitudeUTM")

ebd <- ebd[longitude %in% ebd_locs$longitudeWGS & latitude %in% ebd_locs$latitudeWGS, ]
```

202 4.4 Prepare species of interest

```r
# read in species list
specieslist <- read.csv("data/species_list.csv")

# set species of interest
soi <- specieslist$scientific_name

ebdSpSum <- ebd[, .(
  nSp = .N,
  totSoiSeen = length(intersect(scientific_name, soi))
),
by = list(sampling_event_identifier)
]

# write to file and link with checklist id later
fwrite(ebdSpSum, file = "data/03_data-nspp-per-chk.csv")
```

4.5 Prepare checklists for observer score

```r
# 1. add new columns of decimal time and julian date
ebd[, `:=`(
  decimalTime = time_to_decimal(time_observations_started),
  julianDate = yday(as.POSIXct(observation_date))
)]

ebdEffChk <- setDF(ebd) %>%
  mutate(year = year(observation_date)) %>%
  distinct(
    sampling_event_identifier, observer_id,
    year,
    duration_minutes, effort_distance_km, effort_area_ha,
    longitude, latitude,
    locality, locality_id,
    decimalTime, julianDate, number_observers
  ) %>%
  # drop rows with NAs in cols used in the model
  tidyr::drop_na(
    sampling_event_identifier, observer_id,
    duration_minutes, decimalTime, julianDate
  ) %>%
  # drop years below 2013
  filter(year >= 2013)

# 3. join to covariates and remove large groups (> 10)
ebdChkSummary <- inner_join(ebdEffChk, ebdSpSum)

# remove ebird data
rm(ebd)
gc()
```

4.6 Get landcover

Read in land cover type data resampled at 1km resolution.

```r
# read in 1km landcover and set 0 to NA
library(raster)
landcover <- raster::raster("data/landUseClassification/lc_01000m.tif")
landcover[landcover == 0] <- NA

# get locs in utm coords
locs <- distinct(
  ebdChkSummary, sampling_event_identifier, longitude, latitude,
  locality, locality_id
)
locs <- st_as_sf(locs, coords = c("longitude", "latitude")) %>%
  `st_crs<-`(4326) %>%
  st_transform(32643) %>%
  st_coordinates()

# get for unique points
landcoverVec <- raster::extract(
  x = landcover,
  y = locs
)

# assign to df and overwrite
setDT(ebdChkSummary)[, landcover := landcoverVec]
```

4.7 Filter checklist data 206

```
# change names for easy handling
setnames(ebdChkSummary, c(
  "locality",
  "locality_id", "latitude", "longitude", "observer", "sei",
  "duration", "distance", "area", "nObs", "decimalTime", "julianDate",
  "year", "nSp", "nSoi", "landcover"
))

# count data points per observer
obscount <- count(ebdChkSummary, observer) %>%
  filter(n >= 10)

# make factor variables and remove obs not in obscount
# also remove 0 durations
ebdChkSummary <- ebdChkSummary %>%
  mutate(
    distance = ifelse(is.na(distance), 0, distance),
    duration = if_else(is.na(duration), 0.0, as.double(duration))
  ) %>%
  filter(
    observer %in% obscount$observer,
    duration > 0,
    duration <= 300,
    nSoi >= 0,
    distance <= 5,
    !is.na(nSoi)
  ) %>%
  mutate(
    landcover = as.factor(landcover),
    observer = as.factor(observer)
  ) %>%
  drop_na(landcover)

# editing julian date to model it in a linear fashion
unique(ebdChkSummary$julianDate)

ebdChkSummary <- ebdChkSummary %>%
  mutate(
    newjulianDate =
      case_when(
        julianDate >= 334 & julianDate <= 365 ~ (julianDate - 333),
        julianDate >= 1 & julianDate <= 152 ~ (julianDate + 31)
      )
  ) %>%
  drop_na(newjulianDate)

# save to file for later reuse
fwrite(ebdChkSummary, file = "data/03_data-covars-perChklist.csv")
```

4.8 Model observer expertise

Our observer expertise model aims to include the random intercept effect of observer identity, with a random slope effect of duration. This models the different rate of species accumulation by different observers, as well as their different starting points.

```
# uses either a subset or all data
library(lmerTest)

# here we specify a glmm with random effects for observer
# time is considered a fixed log predictor and a random slope
modObsExp <- glmer(nSoi ~ sqrt(duration) +
  landcover +
  sqrt(decimalTime) +
  I((sqrt(decimalTime))^2) +
  log(newjulianDate) +
  I((log(newjulianDate)^2)) +
  (1 | observer) + (0 + duration | observer),
data = ebdChkSummary, family = "poisson"
)

# make dir if absent
if (!dir.exists("data/modOutput")) {
  dir.create("data/modOutput")
}

# write model output to text file
{
  writeLines(R.utils::captureOutput(list(Sys.time(), summary(modObsExp))),
    con = "data/modOutput/03_model-output-expertise.txt"
  )
}

# make df with means
observer <- unique(ebdChkSummary$observer)

# predict at 60 mins on the most common landcover (deciduous forests)
dfPredict <- ebdChkSummary %>%
  summarise_at(vars(duration, decimalTime, newjulianDate), list(~ mean(.))) %>%
  mutate(duration = 60, landcover = as.factor(2)) %>%
  tidyr::crossing(observer)

# run predict from model on it
dfPredict <- mutate(dfPredict,
  score = predict(modObsExp,
    newdata = dfPredict,
    type = "response",
    allow.new.levels = TRUE
  )
) %>%
  mutate(score = scales::rescale(score))

fwrite(dfPredict %>% dplyr::select(observer, score),
  file = "data/03_data-obsExpertise-score.csv"
)
```

5 Prepare Nearest Neighbour and Road Metrics

The goal of this section is to determine how far each checklist location is from the nearest road, and how far each site is from its nearest neighbour. This involves finding the pairwise distance between a large number of unique checklist locations to a vast number of roads, as well as to each other.

This section is implemented in Python 3, because we want to use `scipy`'s efficient spatial indexing functions based on "kDtrees".

5.1 Prepare libraries

Here, we import required libraries — you may need to install these for your system.

```
# import basic python libs
import itertools
from operator import itemgetter
import numpy as np
import matplotlib.pyplot as plt
import math

# libs for dataframes
import pandas as pd

# import libs for geodata
from shapely.ops import nearest_points
import geopandas as gpd

# import ckdtree
from scipy.spatial import cKDTree
from shapely.geometry import Point, MultiPoint, LineString, MultiLineString
```

5.2 Prepare data for processing

First we read in the roads shapefile, which is obtained from the Open Street Map database. Then we read in the checklist covariates, and extract the unique coordinate pairs. All data are reprojected to be in the UTM 43N coordinate system.

We define a custom Python function to separate multi-feature geometries (here, roads which are in parts) into single feature geometries. Then we define a function to use the K-dimensional trees method from `scipy` to find the distance between two geometries, here, the distance between the locations and the nearest road. We define another function to find the distance between checklist locations and all other checklist locations.

We use these functions to find the distance between each checklist location and the nearest road and the next nearest site.

5.3 Read in data

Here we read in the data and convert it to a spatial format.

```python
# read in roads shapefile
roads = gpd.read_file("data/spatial/roads_studysite_2019/roads_studysite_2019.shp")
roads.head()
```

```python
# read in checklist covariates for conversion to gpd
# get unique coordinates, assign them to the df
# convert df to geo-df
chkCovars = pd.read_csv("data/03_data-covars-perChklist.csv")
unique_locs = chkCovars.drop_duplicates(subset=['longitude',
                                         'latitude'])[['longitude', 'latitude']]
unique_locs['coordId'] = np.arange(1, unique_locs.shape[0]+1)
chkCovars = chkCovars.merge(unique_locs, on=['longitude', 'latitude'])

unique_locs = gpd.GeoDataFrame(
unique_locs,
geometry = gpd.points_from_xy(unique_locs.longitude, unique_locs.latitude))
unique_locs.crs = {'init' :'epsg:4326'}

# reproject spatials to 43n epsg 32643

roads = roads.to_crs({'init': 'epsg:32643'})
unique_locs = unique_locs.to_crs({'init': 'epsg:32643'})
```

5.4 Define functions for geometry simplication and distances

```python
# function to simplify multilinestrings
def simplify_roads(complex_roads):
    simpleRoads = []
    for i in range(len(complex_roads.geometry)):
        feature = complex_roads.geometry.iloc[i]
        if feature.geom_type == "LineString":
            simpleRoads.append(feature)
        elif feature.geom_type == "MultiLineString":
            for road_level2 in feature:
                simpleRoads.append(road_level2)
    return simpleRoads

# function to use ckdtrees to find the nearest road
def ckdnearest(gdfA, gdfB):
    A = np.concatenate(
    [np.array(geom.coords) for geom in gdfA.geometry.to_list()])
    simplified_features = simplify_roads(gdfB)
    B = [np.array(geom.coords) for geom in simplified_features]
    B = np.concatenate(B)
    ckd_tree = cKDTree(B)
    dist, idx = ckd_tree.query(A, k=1)
    return dist

# function to use ckdtrees for nearest other checklist point
def ckdnearest_point(gdfA, gdfB):
    A = np.concatenate(
    [np.array(geom.coords) for geom in gdfA.geometry.to_list()])
    #simplified_features = simplify_roads(gdfB)
    B = np.concatenate(
    [np.array(geom.coords) for geom in gdfB.geometry.to_list()])
    #B = np.concatenate(B)
    ckd_tree = cKDTree(B)
    dist, idx = ckd_tree.query(A, k=[2])
    return dist
```

5.5 Get distance from sites to road and each other

```
# get distance to nearest road
unique_locs['dist_road'] = ckdnearest(unique_locs, roads)

# get distance to nearest other site
unique_locs['nnb'] = ckdnearest_point(unique_locs, unique_locs)
```

```
# check for added columns
unique_locs.head()
```

```
# write to file
unique_locs = pd.DataFrame(unique_locs.drop(columns='geometry'))
unique_locs['dist_road'] = unique_locs['dist_road']
unique_locs['nnb'] = unique_locs['nnb']
unique_locs.to_csv(path_or_buf = "data/locs_dist_to_road.csv", index=False)

# merge unique locs with chkCovars
chkCovars = chkCovars.merge(unique_locs, on=['latitude', 'longitude', 'coordId'])
```

```
# check that metrics have been added to the data
chkCovars.head()
```

```
# save data to local file
# overwrite pre-existing data
chkCovars.to_csv("data/03_data-covars-perChklist.csv")
```

6 Examining Spatial Sampling Bias

The goal of this section is to show how far each checklist location is from the nearest road, and how far each site is from its nearest neighbour. This follows finding the pairwise distance between a large number of unique checklist locations to a vast number of roads, as well as to each other.

6.1 Prepare libraries

```
# load libraries
# for data
library(sf)
library(rnaturalearth)
library(dplyr)
library(readr)
library(purrr)

# for plotting
library(scales)
library(ggplot2)
library(ggspatial)
library(scico)

# round any function
```

```
round_any <- function(x, accuracy = 20000) {
  round(x / accuracy) * accuracy
}
# ci function
ci <- function(x) {
  qnorm(0.975) * sd(x, na.rm = TRUE) / sqrt(length(x))
}
```

6.2 Read checklist data

Read in checklist data with distance to nearest neighbouring site, and the distance to the nearest road.

```
# read from local file
chkCovars <- read_csv("data/03_data-covars-perChklist.csv")
```

6.2.1 Spatially explicit filter on checklists

We filter the checklists by the boundary of the study area. This is *not* the extent.

```
chkCovars <- st_as_sf(chkCovars, coords = c("longitude", "latitude")) %>%
  `st_crs<-`(4326) %>%
  st_transform(32643)

# read wg
wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
  st_transform(32643)
# get bounding box
bbox <- st_bbox(wg)

# spatial subset
chkCovars <- chkCovars %>%
  mutate(id = 1:nrow(.)) %>%
  filter(id %in% unlist(st_contains(wg, chkCovars)))
```

6.2.2 Get background land for plotting

```
# add land
land <- ne_countries(
  scale = 50, type = "countries", continent = "asia",
  country = "india",
  returnclass = c("sf")
) %>%
  st_transform(32643)

# add roads data
roads <- st_read("data/spatial/roads_studysite_2019/roads_studysite_2019.shp") %>%
  st_transform(32643)
```

6.3 Prepare Main Text Figure 3

6.3.1 Prepare histogram of distance to roads

Figure code is hidden in versions rendered as HTML or PDF.

6.3.2 Table: Distance to roads

```r
# write the mean and ci95 to file
chkCovars %>%
  st_drop_geometry() %>%
  select(dist_road, nnb) %>%
  tidyr::pivot_longer(
    cols = c("dist_road", "nnb"),
    names_to = "variable"
  ) %>%
  group_by(variable) %>%
  summarise_at(
    vars(value),
    list(~ mean(.), ~ sd(.), ~ min(.), ~ max(.))
  ) %>%
  write_csv("data/results/distance_roads_sites.csv")
```

6.3.3 Distance to nearest neighbouring site

```r
# get unique locations from checklists
locs_unique <- cbind(
  st_drop_geometry(chkCovars),
  st_coordinates(chkCovars)
) %>%
  as_tibble()

locs_unique <- distinct(locs_unique, X, Y, .keep_all = T)
```

Figure code is hidden in versions rendered as HTML and PDF.

6.3.4 Spatial distribution of distances to neighbours

6.4 Main Text Figure 3

```r
# get locations
points <- chkCovars %>%
  bind_cols(as_tibble(st_coordinates(.))) %>%
  st_drop_geometry() %>%
  mutate(X = round_any(X, 2500), Y = round_any(Y, 2500))

# count points
points <- count(points, X, Y)
```

Figure code is hidden in versions rendered as HTML and PDF.

7 Checking Temporal Sampling Frequency

How often are checklists recorded in each grid cell?

7.1 Load libraries

```r
# load libraries
library(tidyverse)
library(sf)

# for plotting
```

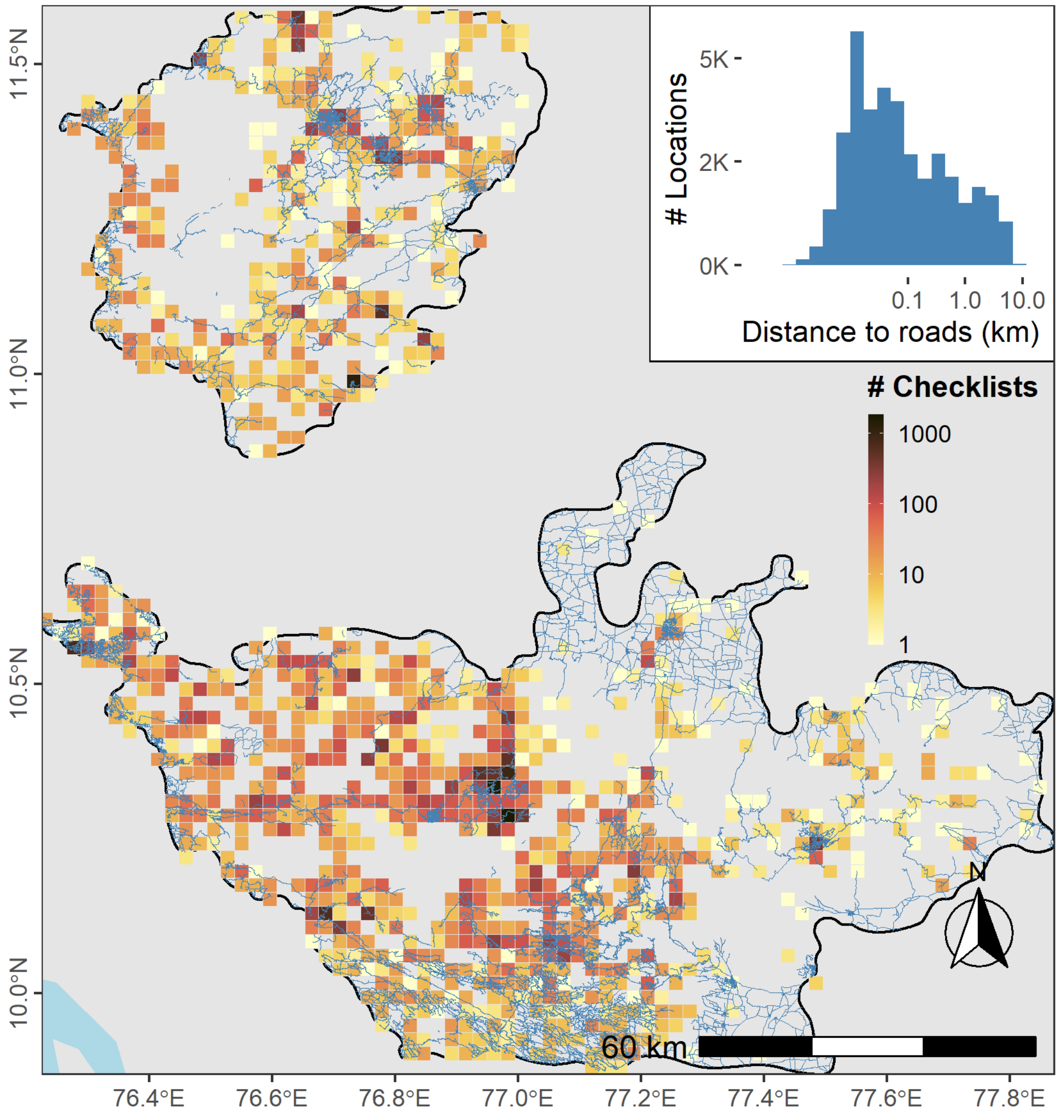

Figure 3: Spatial sampling bias of eBird observations across the Nilgiri and the Anamalai hills. A large proportion of localities/sites were next to roads and were on average only ~300m from another locality/site. Each cell here is 2.5km x 2.5km

```r
library(ggplot2)
library(scico)
library(ggthemes)
library(ggspatial)
```

7.2 Load checklist data

Here we load filtered checklist data and convert to UTM 43N coordinates.

```r
# load checklist data
load("data/01_ebird_data_prelim_processing.rdata")

# get checklists
data <- distinct(
  dataGrouped, sampling_event_identifier, observation_date,
  longitude, latitude
)

# remove old data
rm(dataGrouped)

# transform to UTM 43N
data <- st_as_sf(data, coords = c("longitude", "latitude"), crs = 4326)
data <- st_transform(data, crs = 32643)

# get coordinates and bind to data
data <- cbind(
  st_drop_geometry(data),
  st_coordinates(data)
)

# bin to 1000m
data <- mutate(data,
  X = plyr::round_any(X, 2500),
  Y = plyr::round_any(Y, 2500)
)
```

7.3 Get time differences per grid cell

```r
# get time differences in days
data <- mutate(data, observation_date = as.POSIXct(observation_date))
data <- nest(data, data = c("sampling_event_identifier", "observation_date"))

# map over data
data <- mutate(data,
  lag_metrics = lapply(data, function(df) {
    df <- arrange(df, observation_date)

    lag <- as.numeric(diff(df$observation_date, na.rm = TRUE) / (24 * 3600))

    data <- tibble(
      mean_lag = mean(lag, na.rm = TRUE),
      median_lag = median(lag, na.rm = TRUE),
      sd_lag = sd(lag, na.rm = TRUE),
      n_chk = nrow(df)
    )

    data
  })
)

# unnest lag metrics
data_lag <- select(data, -data)
data_lag <- unnest(data_lag, cols = "lag_metrics")

# set the mean and median to infinity if nchk is 1
data_lag <- mutate(data_lag,
  mean_lag = ifelse(n_chk == 1, Inf, mean_lag),
  median_lag = ifelse(n_chk == 1, Inf, median_lag),
  sd_lag = ifelse(n_chk == 1, Inf, sd_lag)
)

# set all 0 to 1
data_lag <- mutate(data_lag,
  mean_lag = mean_lag + 1,
  median_lag = median_lag + 1
)
# melt data by tile
# data_lag = pivot_longer(data_lag, cols = c("mean_lag", "median_lag", "sd_lag"))
```

7.4 Time Since Previous Checklist

7.4.1 Get aux data

```r
# hills data
wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
  st_transform(32643)

roads <- st_read("data/spatial/roads_studysite_2019/roads_studysite_2019.shp") %>%
  st_transform(32643)

# add land
library(rnaturalearth)
land <- ne_countries(
  scale = 50, type = "countries", continent = "asia",
  country = "india",
  returnclass = c("sf")
) %>%
  st_transform(32643)

bbox <- st_bbox(wg)
```

7.4.2 Histogram of lags

```r
# get lags
data <- mutate(data,
  lag_hist = lapply(data, function(df) {
    df <- arrange(df, observation_date)

    lag <- as.numeric(diff(df$observation_date, na.rm = TRUE) / (24 * 3600))
    data <- tibble(
      lag = lag + 1,
      index = seq(lag)
    )

    data
  })
)

# unnest lags
data_hist <- select(data, X, Y, lag_hist) %>%
  unnest(cols = "lag_hist")

fig_hist_lag <-
  ggplot(data_hist) +
  geom_histogram(
    aes(x = lag),
    bins = 10, size = 0.2, fill = "steelblue"
  ) +
  scale_x_log10() +
  scale_y_continuous(
    labels = scales::label_number(
      scale = 0.001, accuracy = 1,
      suffix = "K"
    ),
    limits = c(0, 10.5e3)
  ) +
  theme_few() +
  theme(
    plot.background = element_rect(fill = "white", colour = 1),
    panel.background = element_blank(),
    panel.border = element_blank(), axis.line = element_blank(),
    axis.text.y = element_text(
      angle = 90,
      hjust = 0.5
    )
  ) +
  labs(
    x = "Days since prev. checklist",
    y = "# checklists"
  )

# make plot
fig_lag_spatial <-
  ggplot(data_lag) +
  geom_sf(data = land, fill = "grey90", col = NA) +
  geom_sf(
    data = wg,
    fill = NA
  ) +
  annotation_custom(
    grob = fig_hist_lag %>% ggplotGrob(),
    xmin = bbox["xmax"] - (bbox["xmax"] - bbox["xmin"]) / 2.5,
    xmax = bbox["xmax"],
```

```r
    ymin = bbox["ymax"] - (bbox["ymax"] - bbox["ymin"]) / 3,
    ymax = bbox["ymax"]
  ) +
  geom_tile(
    aes(X, Y,
      fill = mean_lag
    ),
    col = "grey90"
  ) +
  geom_sf(
    data = roads, size = 0.2, col = "indianred"
  ) +
  scale_fill_scico(
    palette = "oslo",
    direction = 1,
    trans = "log10",
    na.value = alpha("gold")
  ) +
  annotation_north_arrow(
    location = "br", which_north = "true",
    pad_x = unit(0.1, "in"), pad_y = unit(0.5, "in"),
    style = north_arrow_fancy_orienteering
  ) +
  annotation_scale(location = "br", width_hint = 0.4, text_cex = 1) +
  coord_sf(
    expand = FALSE,
    xlim = bbox[c("xmin", "xmax")],
    ylim = bbox[c("ymin", "ymax")]
  ) +
  ggthemes::theme_few() +
  theme(
    legend.position = c(0.9, 0.53),
    legend.background = element_blank(),
    legend.key = element_rect(fill = "grey90"),
    axis.text.y = element_text(
      angle = 90,
      hjust = 0.5
    ),
    axis.title = element_blank(),
    panel.background = element_rect(fill = "lightblue")
  ) +
  labs(
    fill = "Mean days\nb/w checklists"
  )

# save figure
ggsave(
  filename = "figs/fig_lag_spatial.png", width = 6.5, height = 6.5,
  device = png()
)
```

7.5 Checklists per Month

```r
# get two week period by date
data <- select(data, X, Y, data)
```

```r
# unnest
data <- unnest(data, cols = "data")

# get fortnight
library(lubridate)
data <- mutate(data,
  week = week(observation_date),
  week = plyr::round_any(week, 2),
  year = year(observation_date),
  month = month(observation_date)
)

# count checklists per fortnight
data_count <- count(data, month, year)

ggplot(data_count) +
  geom_boxplot(
    aes(
      x = factor(month),
      y = n
    ),
    fill = "steelblue"
  ) +
  scale_y_log10(
    limits = c(10, NA)
  ) +
  theme_classic() +
  theme(
    axis.text.y = element_text(
      angle = 90,
      hjust = 0.5
    )
  ) +
  labs(
    x = "Month",
    y = "# checklists"
  )

ggsave(filename = "figs/fig_chk_per_month.png")
```

8 Adding Covariates to Checklist Data

In this section, we prepare a final list of covariates, after taking into account spatial sampling bias (examined in the previous section), temporal bias and observer expertise scores.

8.1 Prepare libraries and data

```r
# load libs for data
library(dplyr)
library(readr)
library(stringr)
library(purrr)
library(glue)
library(tidyr)

# check for velox and install
library(devtools)
if (!"velox" %in% installed.packages()) {
  install_github("hunzikp/velox")
}

# load spatial
library(raster)
library(rgeos)
library(velox)
library(sf)

# load saved data object
load("data/01_data_prelim_processing.rdata")
```

8.2 Spatial subsampling

Sampling bias can be introduced into citizen science due to the often ad-hoc nature of data collection (Sullivan et al. 2014). For eBird, this translates into checklists reported when convenient, rather than at regular or random points in time and space, leading to non-independence in the data if observations are spatio-temporally clustered (Johnston et al. 2019). Spatio-temporal autocorrelation in the data can be reduced by sub-sampling at an appropriate spatial resolution, and by avoiding temporal clustering. We estimated two simple measures of spatial clustering: the distance from each site to the nearest road (road data from OpenStreetMap; (OpenStreetMap contributors 2019)), and the nearest-neighbor distance for each site. Sites were strongly tied to roads (mean distance to road ± SD = 390.77 ± 859.15 m; range = 0.28 m – 7.64 km) and were on average only 297 m away from another site (SD = 553 m; range = 0.14 m – 12.85 km) (Figure 3). This analysis was done in the previous section.

Here, to further reduce spatial autocorrelation, we divided the study area into a grid of 1km wide square cells and picked checklists from one site at random within each grid cell.

Prior to running this analysis, we checked how many checklists/data would be retained given a particular value of distance to account for spatial independence. This analysis can be accessed in Section 8 of the Supplementary Material. We show that over 80% of checklists are retained with a distance cutoff of 1km. In addition, a number of thinning approaches were tested to determine which method retained the highest proportion of points, while accounting for sampling effort (time and distance). This analysis can be accessed in Section 9 of the Supplementary Material.

```r
# grid based spatial thinning
gridsize <- 1000 # grid size in metres
effort_distance_max <- 1000 # removing checklists with this distance

# make grids across the study site
hills <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
  st_transform(32643)
grid <- st_make_grid(hills, cellsize = gridsize)

# split data by species
data_spatial_thin <- split(x = dataGrouped, f = dataGrouped$scientific_name)

# spatial thinning on each species retains
# site with maximum visits per grid cell
data_spatial_thin <- map(data_spatial_thin, function(df) {

  # count visits per locality
  df <- group_by(df, locality) %>%
    mutate(tot_effort = length(sampling_event_identifier)) %>%
    ungroup()

  # remove sites with distances above spatial independence
  df <- df %>%
    filter(effort_distance_km <= effort_distance_max) %>%
    st_as_sf(coords = c("longitude", "latitude")) %>%
    `st_crs<-`(4326)

  # transform to regional UTM 43N and add id
  df <- df %>%
    st_transform(32643) %>%
    mutate(coordId = 1:nrow(.)) %>%
    bind_cols(as_tibble(st_coordinates(.)))

  # whcih cell has which coords
  grid_overlap <- st_contains(grid, df) %>%
    unclass() %>%
    discard(.p = is_empty)

  # count length of grid overlap list
  # this is the number of cells with points in them
  sampled_cells <- length(grid_overlap)

  # make tibble
  grid_overlap <- tibble(
    uid_cell = seq(length(grid_overlap)), # the uid_cell is specific to this sp.
    coordId = grid_overlap
  )

  # unnest
  grid_overlap <- unnest(grid_overlap, cols = "coordId")

  # join grid cell overlap with coordinate data
  df <- left_join(df,
    grid_overlap,
    by = "coordId"
  ) %>%
    st_drop_geometry()

  # for each uid_cell, select coord where effort is max
  points_max <- df %>%
    group_by(uid_cell) %>%
    filter(tot_effort == max(tot_effort)) %>%
    # there may be multiple rows with max effort, select first
    filter(row_number() == 1)

  # check for number of samples
  assertthat::assert_that(
    assertthat::are_equal(sampled_cells, nrow(points_max),
      msg = "spatial thinning error: more samples than\\
                          sampled cells"
    )
  )
```

```
  # check that there is one sample per cell
  assertthat::assert_that(
    assertthat::are_equal(
      max(count(points_max, uid_cell)$n), 1
    )
  )

  points_max
})

# remove old data
rm(dataGrouped)
```

8.3 Temporal subsampling

Additionally, from each selected site, we randomly selected a maximum of 10 checklists, which reduced temporal autocorrelation.

```
# subsample data for random 10 observations
dataSubsample <- map(data_spatial_thin, function(df) {
  df <- ungroup(df)
  df_to_locality <- split(x = df, f = df$locality)
  df_samples <- map_if(
    .x = df_to_locality,
    .p = function(x) {
      nrow(x) > 10
    },
    .f = function(x) sample_n(x, 10, replace = FALSE)
  )

  bind_rows(df_samples)
})

# bind all rows for data frame
dataSubsample <- bind_rows(dataSubsample)

# remove previous data
rm(data_spatial_thin)
```

8.4 Add checklist calibration index

Load the CCI computed in the previous section. The CCI was the lone observer's expertise score for single-observer checklists, and the highest expertise score among observers for group checklists.

```
# read in obs score and extract numbers
expertiseScore <- read_csv("data/03_data-obsExpertise-score.csv") %>%
  mutate(numObserver = str_extract(observer, "\\d+")) %>%
  dplyr::select(-observer)

# group seis consist of multiple observers
# in this case, seis need to have the highest expertise observer score
# as the associated covariate

# get unique observers per sei
dataSeiScore <- distinct(
  dataSubsample, sampling_event_identifier,
  observer_id
) %>%
  # make list column of observers
  mutate(observers = str_split(observer_id, ",")) %>%
  unnest(cols = c(observers)) %>%
  # add numeric observer id
  mutate(numObserver = str_extract(observers, "\\d+")) %>%
  # now get distinct sei and observer id numeric
  distinct(sampling_event_identifier, numObserver)

# now add expertise score to sei
dataSeiScore <- left_join(dataSeiScore, expertiseScore,
  by = "numObserver"
) %>%
  # get max expertise score per sei
  group_by(sampling_event_identifier) %>%
  summarise(expertise = max(score))

# add to dataCovar
dataSubsample <- left_join(dataSubsample, dataSeiScore,
  by = "sampling_event_identifier"
)

# remove data without expertise score
dataSubsample <- filter(dataSubsample, !is.na(expertise))
```

8.5 Add climatic and landscape covariates

Reload climate and land cover predictors prepared previously.

```
# list landscape covariate stacks
landscape_files <- "data/spatial/landscape_resamp01_km.tif"

# read in as stacks
landscape_data <- stack(landscape_files)

# get proper names
elev_names <- c("elev", "slope", "aspect")
chelsa_names <- c("bio15a", "bio4a")

names(landscape_data) <- as.character(glue('{c(elev_names, chelsa_names, "landcover")}'))
```

8.6 Spatial buffers around selected checklists

Every checklist on eBird is associated with a latitude and longitude. However, the coordinates entered by an observer may not accurately depict the location at which a species was detected. This can occur for two reasons: first, traveling checklists are often associated with a single location along the route travelled by observers; and second, checklist locations could be assigned to a 'hotspot' – a location that is marked by eBird as being frequented by multiple observers. In many cases, an observation might be assigned to a hotspot even though the observation was not made at the precise location of the hotspot (J. 2017). Johnston et al., (2019) showed that a large proportion of observations occurred within a 3km grid, even for those checklists up to 5km in length. Hence to adjust for spatial precision, we considered a minimum radius of 2.5km around each unique locality when sampling environmental covariate values.

```
# assign neighbourhood radius in m
sample_radius <- 2.5 * 1e3

# get distinct points and make buffer
ebird_buff <- dataSubsample %>%
  ungroup() %>%
  distinct(X, Y) %>%
  mutate(id = 1:nrow(.)) %>%
  crossing(sample_radius) %>%
  arrange(id) %>%
  group_by(sample_radius) %>%
  nest() %>%
  ungroup()


# convert to spatial features
ebird_buff <- mutate(ebird_buff,
  data = map2(
    data, sample_radius,
    function(df, rd) {
      df_sf <- st_as_sf(df, coords = c("X", "Y"), crs = 32643) %>%
        # add long lat
        bind_cols(as_tibble(st_coordinates(.))) %>%
        # rename(longitude = X, latitude = Y) %>%
        # # transform to modis projection
        # st_transform(crs = 32643) %>%
        # buffer to create neighborhood around each point
        st_buffer(dist = rd)
    }
  )
)
```

8.7 Spatial buffer-wide covariates

8.7.1 Mean climatic covariates

All climatic covariates are sampled by considering the mean values within a 2.5km radius as discussed above and prefixed “am_”.

```
# get area mean for all preds except landcover, which is the last one
env_area_mean <- purrr::map(ebird_buff$data, function(df) {
  stk <- landscape_data[[-dim(landscape_data)[3]]] # removing landcover here
  velstk <- velox(stk)
  dextr <- velstk$extract(
    sp = df, df = TRUE,
    fun = function(x) mean(x, na.rm = T)
  )

  # assign names for joining
  names(dextr) <- c("id", names(stk))
  return(as_tibble(dextr))
})

# join to buffer data
ebird_buff <- ebird_buff %>%
```

```r
  mutate(data = map2(data, env_area_mean, inner_join, by = "id"))
```

8.7.2 Proportions of land cover type

All land cover covariates were sampled by considering the proportion of each land cover type within a 2.5km radius.

```r
# get the last element of each stack from the list
# this is the landcover at that resolution
lc_area_prop <- purrr::map(ebird_buff$data, function(df) {
  lc <- landscape_data[[dim(landscape_data)[3]]] # accessing landcover here
  lc_velox <- velox(lc)
  lc_vals <- lc_velox$extract(sp = df, df = TRUE)
  names(lc_vals) <- c("id", "lc")

  # get landcover proportions
  lc_prop <- count(lc_vals, id, lc) %>%
    group_by(id) %>%
    mutate(
      lc = glue('lc_{str_pad(lc, 2, pad = "0")}'),
      prop = n / sum(n)
    ) %>%
    dplyr::select(-n) %>%
    tidyr::pivot_wider(
      names_from = lc,
      values_from = prop,
      values_fill = list(prop = 0)
    ) %>%
    ungroup()

  return(lc_prop)
})

# join to data
ebird_buff <- ebird_buff %>%
  mutate(data = map2(data, lc_area_prop, inner_join, by = "id"))
```

8.7.3 Link environmental covariates to checklists

```r
# duplicate scale data
data_at_scale <- ebird_buff

# join the full data to landscape samples at each scale
data_at_scale$data <- map(data_at_scale$data, function(df) {
  df <- st_drop_geometry(df)
  df <- inner_join(dataSubsample, df, by = c("X", "Y"))
  return(df)
})
```

Save data to file.

```r
# write to file
pmap(data_at_scale, function(sample_radius, data) {
  write_csv(data, path = glue('data/04_data-covars-{str_pad(sample_radius/1e3, 2, pad = "0")}km.csv'))
  message(glue('export done: data/04_data-covars-{str_pad(sample_radius/1e3, 2, pad = "0")}km.csv'))
})
```

9 Modelling Species Occupancy

9.0.1 Load necessary libraries

```r
# Load libraries
library(auk)
library(lubridate)
library(sf)
library(unmarked)
library(raster)
library(ebirdst)
library(MuMIn)
library(AICcmodavg)
library(fields)
library(tidyverse)
library(doParallel)
library(snow)
library(openxlsx)
library(data.table)
library(dplyr)
library(ecodist)

# Source necessary functions
source("R/fun_screen_cor.R")
source("R/fun_model_estimate_collection.r")
```

9.1 Load dataframe and scale covariates

Here, we load the required dataframe that contains 10 random visits to a site and environmental covariates that were prepared at a spatial scale of 2.5 sq.km. We also scaled all covariates (mean around 0 and standard deviation of 1). Next, we ensured that only Traveling and Stationary checklists were considered. Even though stationary counts have no distance traveled, we defaulted all stationary accounts to an effective distance of 100m, which we consider the average maximum detection radius for most bird species in our area. Following this, we excluded predictors with a Pearson coefficient > 0.5.

Please note that species-specific plots of probabilities of occupancy as a function of environmental data can be accessed in Section 10 of the Supplementary Material.

```r
# Load in the prepared dataframe that contains 10 random visits to each site
dat <- fread("data/04_data-covars-2.5km.csv", header = T)
setDF(dat)
head(dat)

# Some more pre-processing to get the right data structures

# Ensuring that only Traveling and Stationary checklists were considered
names(dat)
dat <- dat %>% filter(protocol_type %in% c("Traveling", "Stationary"))

# We take all stationary counts and give them a distance of 100 m (so 0.1 km),
# as that's approximately the max normal hearing distance for people doing point
# counts.
dat <- dat %>%
  mutate(effort_distance_km = replace(
    effort_distance_km,
    which(effort_distance_km == 0 &
      protocol_type == "Stationary"),
    0.1
  ))

# Converting time observations started to numeric and adding it as a new column
# This new column will be minute_observations_started
dat <- dat %>%
  mutate(
    min_obs_started = strtoi(
      as.difftime(
        time_observations_started,
        format = "%H:%M:%S", units = "mins"
      )
    )
  )

# Adding the julian date to the dataframe
dat <- dat %>%
  mutate(julian_date = lubridate::yday(dat$observation_date))

# recode julian date to model it as a linear predictor
dat <- dat %>%
  mutate(
    newjulianDate =
      case_when(
        (julian_date >= 334 & julian_date) <= 365 ~
        (julian_date - 333),
        (julian_date >= 1 & julian_date) <= 152 ~
        (julian_date + 31)
      )
  ) %>%
  drop_na(newjulianDate)

# recode time observations started to model it as a linear predictor
dat <- dat %>%
  mutate(
    newmin_obs_started = case_when(
      min_obs_started >= 300 & min_obs_started <= 720 ~
      abs(min_obs_started - 720),
      min_obs_started >= 720 & min_obs_started <= 1140 ~
      abs(720 - min_obs_started)
    )
  ) %>%
  drop_na(newmin_obs_started)

# Removing other unnecessary columns from the dataframe and creating a clean one without the rest
names(dat)
dat <- dat[, -c(1, 4, 5, 16, 18, 21, 23, 25, 26, 36, 37)]

# Rename column names:
names(dat) <- c(
  "duration_minutes", "effort_distance_km", "locality",
  "locality_type", "locality_id", "observer_id",
  "observation_date", "scientific_name", "observation_count",
  "protocol_type", "number_observers", "pres_abs", "tot_effort",
  "longitude", "latitude", "expertise", "elev", "bio15a.y", "bio4a.y",
  "lc_01.y", "lc_02.y", "lc_05.y", "lc_04.y", "lc_09.y",
  "lc_07.y", "lc_03.y", "julian_date", "min_obs_started"
)

dat.1 <- dat %>%
  mutate(
    year = year(observation_date),
    pres_abs = as.integer(pres_abs)
  ) # occupancy modeling requires an integer response

# Scaling detection and occupancy covariates
dat.scaled <- dat.1
dat.scaled[, c(1, 2, 11, 16:28)] <- scale(dat.scaled[, c(1, 2, 11, 16:28)]) # Scaling and standardizing detecti
fwrite(dat.scaled, file = "data/05_scaled-covars-2.5km.csv")

# Reload the scaled covariate data
dat.scaled <- fread("data/05_scaled-covars-2.5km.csv", header = T)
setDF(dat.scaled)
head(dat.scaled)

# Ensure observation_date column is in the right format
dat.scaled$observation_date <- format(
  as.Date(
    dat.scaled$observation_date,
    "%m/%d/%Y"
  ),
  "%Y-%m-%d"
)

# Testing for correlations before running further analyses
# Majority are uncorrelated since we decided to keep climatic and land cover predictors and removed elevation.
source("R/fun_screen_cor.R")
names(dat.scaled)
screen.cor(dat.scaled[, c(1, 2, 11, 16:28)], threshold = 0.3)
```

9.2 Running a null model

```r
# All null models are stored in lists below
all_null <- list()

# Add a progress bar for the loop
pb <- txtProgressBar(
  min = 0,
  max = length(unique(dat.scaled$scientific_name)),
  style = 3
) # text based bar

for (i in 1:length(unique(dat.scaled$scientific_name))) {
  data <- dat.scaled %>%
    filter(dat.scaled$scientific_name == unique(dat.scaled$scientific_name)[i])

  # Preparing data for the unmarked model
  occ <- filter_repeat_visits(data,
    min_obs = 1, max_obs = 10,
    annual_closure = FALSE,
    n_days = 3000, # 7 years is considered a period of closure
    date_var = "observation_date",
    site_vars = c("locality_id")
  )

  obs_covs <- c(
    "min_obs_started",
    "duration_minutes",
    "effort_distance_km",
    "number_observers",
    "expertise",
    "julian_date"
  )

  # format for unmarked
  occ_wide <- format_unmarked_occu(occ,
    site_id = "site",
    response = "pres_abs",
    site_covs = c(
      "locality_id", "lc_01.y", "lc_02.y", "lc_05.y",
      "lc_04.y", "lc_09.y", "lc_07.y", "lc_03.y", "bio15a.y", "bio4a.y"
    ),
    obs_covs = obs_covs
  )

  # Convert this dataframe of observations into an unmarked object to start fitting occupancy models
  occ_um <- formatWide(occ_wide, type = "unmarkedFrameOccu")

  # Set up the model
  all_null[[i]] <- occu(~1 ~ 1, data = occ_um)
  names(all_null)[i] <- unique(dat.scaled$scientific_name)[i]
  setTxtProgressBar(pb, i)
}
close(pb)

# Store all the  model outputs for each species
capture.output(all_null, file = "data/results/null_models.csv")
```

9.3 Identifying covariates necessary to model the detection process

Here, we use the unmarked package in R (Fiske and Chandler 2011) to identify detection level covariates that are important for each species. We use AIC criteria to select top models (Burnham, Anderson, and Huyvaert 2011).

```
# All models are stored in lists below
det_dred <- list()

# Subsetting those models whose deltaAIC<2 (Burnham et al., 2011)
top_det <- list()

# Getting model averaged coefficients and relative importance scores
det_avg <- list()
det_imp <- list()
```

```r
# Getting model estimates
det_modelEst <- list()

# Add a progress bar for the loop
pb <- txtProgressBar(
  min = 0,
  max = length(unique(dat.scaled$scientific_name)), style = 3
) # text based bar

for (i in 1:length(unique(dat.scaled$scientific_name))) {
  data <- dat.scaled %>%
    filter(dat.scaled$scientific_name == unique(dat.scaled$scientific_name)[i])

  # Preparing data for the unmarked model
  occ <- filter_repeat_visits(data,
    min_obs = 1, max_obs = 10,
    annual_closure = FALSE,
    n_days = 3000, # 7 years is considered a period of closure
    date_var = "observation_date",
    site_vars = c("locality_id")
  )

  obs_covs <- c(
    "min_obs_started",
    "duration_minutes",
    "effort_distance_km",
    "number_observers",
    "expertise",
    "julian_date"
  )

  # format for unmarked
  occ_wide <- format_unmarked_occu(occ,
    site_id = "site",
    response = "pres_abs",
    site_covs = c(
      "locality_id", "lc_01.y", "lc_02.y", "lc_05.y",
      "lc_04.y", "lc_09.y", "lc_07.y", "lc_03.y", "bio15a.y", "bio4a.y"
    ),
    obs_covs = obs_covs
  )

  # Convert this dataframe of observations into an unmarked object to start fitting occupancy models
  occ_um <- formatWide(occ_wide, type = "unmarkedFrameOccu")

  # Fit a global model with all detection level covariates
  global_mod <- occu(~ min_obs_started +
    julian_date +
    duration_minutes +
    effort_distance_km +
    number_observers +
    expertise ~ 1, data = occ_um)
```

```r
# Set up the cluster
clusterType <- if (length(find.package("snow", quiet = TRUE))) "SOCK" else "PSOCK"
clust <- try(makeCluster(getOption("cl.cores", 6), type = clusterType))

clusterEvalQ(clust, library(unmarked))
clusterExport(clust, "occ_um")

det_dred[[i]] <- pdredge(global_mod, clust)
names(det_dred)[i] <- unique(dat.scaled$scientific_name)[i]

# Get the top models, which we'll define as those with deltaAICc < 2
top_det[[i]] <- get.models(det_dred[[i]], subset = delta < 2, cluster = clust)
names(top_det)[i] <- unique(dat.scaled$scientific_name)[i]

# Obtaining model averaged coefficients
if (length(top_det[[i]]) > 1) {
  a <- model.avg(top_det[[i]], fit = TRUE)
  det_avg[[i]] <- as.data.frame(a$coefficients)
  names(det_avg)[i] <- unique(dat.scaled$scientific_name)[i]

  det_modelEst[[i]] <- data.frame(
    Coefficient = coefTable(a, full = T)[, 1],
    SE = coefTable(a, full = T)[, 2],
    lowerCI = confint(a)[, 1],
    upperCI = confint(a)[, 2],
    z_value = (summary(a)$coefmat.full)[, 3],
    Pr_z = (summary(a)$coefmat.full)[, 4]
  )

  names(det_modelEst)[i] <- unique(dat.scaled$scientific_name)[i]

  det_imp[[i]] <- as.data.frame(MuMIn::importance(a))
  names(det_imp)[i] <- unique(dat.scaled$scientific_name)[i]
} else {
  det_avg[[i]] <- as.data.frame(unmarked::coef(top_det[[i]][[1]]))
  names(det_avg)[i] <- unique(dat.scaled$scientific_name)[i]

  lowDet <- data.frame(lowerCI = confint(top_det[[i]][[1]], type = "det")[, 1])
  upDet <- data.frame(upperCI = confint(top_det[[i]][[1]], type = "det")[, 2])
  zDet <- data.frame(summary(top_det[[i]][[1]])$det[, 3])
  Pr_zDet <- data.frame(summary(top_det[[i]][[1]])$det[, 4])

  Coefficient <- coefTable(top_det[[i]][[1]])[, 1]
  SE <- coefTable(top_det[[i]][[1]])[, 2]

  det_modelEst[[i]] <- data.frame(
    Coefficient = Coefficient[2:length(Coefficient)],
    SE = SE[2:length(SE)],
    lowerCI = lowDet,
    upperCI = upDet,
    z_value = zDet,
    Pr_z = Pr_zDet
  )
```

```
    names(det_modelEst)[i] <- unique(dat.scaled$scientific_name)[i]
  }
  setTxtProgressBar(pb, i)
  stopCluster(clust)
}
close(pb)

## Storing output from the above models in excel sheets

# 1. Store all the model outputs for each species (variable: det_dred() - see above)
write.xlsx(det_dred, file = "data/results/det-dred.xlsx")

# 2. Store all the model averaged outputs for each species and the relative importance score
write.xlsx(det_avg, file = "data/results/det-avg.xlsx", rowNames = T, colNames = T)
write.xlsx(det_imp, file = "data/results/det-imp.xlsx", rowNames = T, colNames = T)

write.xlsx(det_modelEst, file = "data/results/det-modelEst.xlsx", rowNames = T, colNames = T)

# Note if you are unable to write to a file, use (for example)
purrr::map(det_imp, ~ purrr::compact(.)) %>% purrr::keep(~ length(.) != 0)
```

9.4 Land Cover and Climate

Occupancy models estimate the probability of occurrence of a given species while controlling for the probability of detection and allow us to model the factors affecting occurrence and detection independently (Johnston et al. 2018; MacKenzie et al. 2002). The flexible eBird observation process contributes to the largest source of variation in the likelihood of detecting a particular species (Johnston et al. 2019); hence, we included seven covariates that influence the probability of detection for each checklist: ordinal day of year, duration of observation, distance travelled, protocol type, time observations started, number of observers and the checklist calibration index (CCI).

Using a multi-model information-theoretic approach, we tested how strongly our occurrence data fit our candidate set of environmental covariates (Burnham and Anderson 2002). We fitted single-species occupancy models for each species, to simultaneously estimate a probability of detection (p) and a probability of occupancy (ψ) (Fiske and Chandler 2011; MacKenzie et al. 2002). For each species, we fit 256 models, each with a unique combination of the eight (climate and land cover) occupancy covariates and all seven detection covariates (Appendix S5).

Across the 256 models tested for each species, the model with highest support was determined using AICc scores. However, across the majority of the species, no single model had overwhelming support. Hence, for each species, we examined those models which had ΔAICc < 2, as these top models were considered to explain a large proportion of the association between the species-specific probability of occupancy and environmental drivers (Burnham, Anderson, and Huyvaert 2011; Elsen et al. 2017). Using these restricted model sets for each species; we created a model-averaged coefficient estimate for each predictor and assessed its direction and significance (Bartoń 2020). We considered a predictor to be significantly associated with occupancy if the range of the 95% confidence interval around the model-averaged coefficient did not contain zero. Next, we obtained a measure of relative importance of climatic and landscape predictors by calculating cumulative variable importance scores. These scores were calculated by obtaining the sum of model weights (AIC weights) across all models (including the top models) for each predictor across all species.

```
# All models are stored in lists below
lc_clim <- list()

# Subsetting those models whose deltaAIC<2 (Burnham et al., 2011)
top_lc_clim <- list()

# Getting model averaged coefficients and relative importance scores
lc_clim_avg <- list()
lc_clim_imp <- list()
```

```r
# Storing Model estimates
lc_clim_modelEst <- list()

# Add a progress bar for the loop
pb <- txtProgressBar(min = 0, max = length(unique(dat.scaled$scientific_name)), style = 3) # text based bar

for (i in 1:length(unique(dat.scaled$scientific_name))) {
  data <- dat.scaled %>% filter(dat.scaled$scientific_name == unique(dat.scaled$scientific_name)[i])

  # Preparing data for the unmarked model
  occ <- filter_repeat_visits(data,
    min_obs = 1, max_obs = 10,
    annual_closure = FALSE,
    n_days = 3000, # 6 years is considered a period of closure
    date_var = "observation_date",
    site_vars = c("locality_id")
  )

  obs_covs <- c(
    "min_obs_started",
    "duration_minutes",
    "effort_distance_km",
    "number_observers",
    "expertise",
    "julian_date"
  )

  # format for unmarked
  occ_wide <- format_unmarked_occu(occ,
    site_id = "site",
    response = "pres_abs",
    site_covs = c(
      "locality_id", "lc_01.y", "lc_02.y", "lc_05.y",
      "lc_04.y", "lc_09.y", "lc_07.y", "lc_03.y", "bio15a.y", "bio4a.y"
    ),
    obs_covs = obs_covs
  )

  # Convert this dataframe of observations into an unmarked object to start fitting occupancy models
  occ_um <- formatWide(occ_wide, type = "unmarkedFrameOccu")

  model_lc_clim <- occu(~ min_obs_started +
    julian_date +
    duration_minutes +
    effort_distance_km +
    number_observers +
    expertise ~ lc_01.y + lc_02.y + lc_05.y +
    lc_04.y + lc_09.y + lc_07.y + lc_03.y + bio15a.y + bio4a.y, data = occ_um)

  # Set up the cluster
  clusterType <- if (length(find.package("snow", quiet = TRUE))) "SOCK" else "PSOCK"
  clust <- try(makeCluster(getOption("cl.cores", 6), type = clusterType))
```

```r
clusterEvalQ(clust, library(unmarked))
clusterExport(clust, "occ_um")

# Detection terms are fixed
det_terms <- c(
  "p(duration_minutes)", "p(effort_distance_km)", "p(expertise)",
  "p(julian_date)", "p(min_obs_started)",
  "p(number_observers)"
)

lc_clim[[i]] <- pdredge(model_lc_clim, clust, fixed = det_terms)
names(lc_clim)[i] <- unique(dat.scaled$scientific_name)[i]

# Identiying top subset of models based on deltaAIC scores being less than 2 (Burnham et al., 2011)
top_lc_clim[[i]] <- get.models(lc_clim[[i]], subset = delta < 2, cluster = clust)

names(top_lc_clim)[i] <- unique(dat.scaled$scientific_name)[i]

# Obtaining model averaged coefficients for both candidate model subsets
if (length(top_lc_clim[[i]]) > 1) {
  a <- model.avg(top_lc_clim[[i]], fit = TRUE)
  lc_clim_avg[[i]] <- as.data.frame(a$coefficients)
  names(lc_clim_avg)[i] <- unique(dat.scaled$scientific_name)[i]

  lc_clim_modelEst[[i]] <- data.frame(
    Coefficient = coefTable(a, full = T)[, 1],
    SE = coefTable(a, full = T)[, 2],
    lowerCI = confint(a)[, 1],
    upperCI = confint(a)[, 2],
    z_value = (summary(a)$coefmat.full)[, 3],
    Pr_z = (summary(a)$coefmat.full)[, 4]
  )

  names(lc_clim_modelEst)[i] <- unique(dat.scaled$scientific_name)[i]

  lc_clim_imp[[i]] <- as.data.frame(MuMIn::importance(a))
  names(lc_clim_imp)[i] <- unique(dat.scaled$scientific_name)[i]
} else {
  lc_clim_avg[[i]] <- as.data.frame(unmarked::coef(top_lc_clim[[i]][[1]]))
  names(lc_clim_avg)[i] <- unique(dat.scaled$scientific_name)[i]

  lowSt <- data.frame(lowerCI = confint(top_lc_clim[[i]][[1]], type = "state")[, 1])
  lowDet <- data.frame(lowerCI = confint(top_lc_clim[[i]][[1]], type = "det")[, 1])
  upSt <- data.frame(upperCI = confint(top_lc_clim[[i]][[1]], type = "state")[, 2])
  upDet <- data.frame(upperCI = confint(top_lc_clim[[i]][[1]], type = "det")[, 2])
  zSt <- data.frame(z_value = summary(top_lc_clim[[i]][[1]])$state[, 3])
  zDet <- data.frame(z_value = summary(top_lc_clim[[i]][[1]])$det[, 3])
  Pr_zSt <- data.frame(Pr_z = summary(top_lc_clim[[i]][[1]])$state[, 4])
  Pr_zDet <- data.frame(Pr_z = summary(top_lc_clim[[i]][[1]])$det[, 4])

  lc_clim_modelEst[[i]] <- data.frame(
    Coefficient = coefTable(top_lc_clim[[i]][[1]])[, 1],
    SE = coefTable(top_lc_clim[[i]][[1]])[, 2],
    lowerCI = rbind(lowSt, lowDet),
      upperCI = rbind(upSt, upDet),
      z_value = rbind(zSt, zDet),
      Pr_z = rbind(Pr_zSt, Pr_zDet)
    )

    names(lc_clim_modelEst)[i] <- unique(dat.scaled$scientific_name)[i]
  }
  setTxtProgressBar(pb, i)
  stopCluster(clust)
}
close(pb)

# 1. Store all the model outputs for each species (for both landcover and climate)
write.xlsx(lc_clim, file = "data/results/lc-clim.xlsx")

# 2. Store all the model averaged outputs for each species and relative importance scores
write.xlsx(lc_clim_avg, file = "data/results/lc-clim-avg.xlsx", rowNames = T, colNames = T)
write.xlsx(lc_clim_imp, file = "data/results/lc-clim-imp.xlsx", rowNames = T, colNames = T)

# 3. Store all model estimates
write.xlsx(lc_clim_modelEst, file = "data/results/lc-clim-modelEst.xlsx", rowNames = T, colNames = T)
```

9.5 Goodness-of-fit tests

Adequate model fit was assessed using a chi-square goodness-of-fit test using 5000 parametric bootstrap simulations on a global model that included all occupancy and detection covariates (MacKenzie & Bailey, 2004).

```r
goodness_of_fit <- data.frame()

# Add a progress bar for the loop
pb <- txtProgressBar(min = 0, max = length(unique(dat.scaled$scientific_name)), style = 3) # text based bar

for (i in 1:length(unique(dat.scaled$scientific_name))) {
  data <- dat.scaled %>% filter(dat.scaled$scientific_name == unique(dat.scaled$scientific_name)[i])

  # Preparing data for the unmarked model
  occ <- filter_repeat_visits(data,
    min_obs = 1, max_obs = 10,
    annual_closure = FALSE,
    n_days = 3000, # 6 years is considered a period of closure
    date_var = "observation_date",
    site_vars = c("locality_id")
  )

  obs_covs <- c(
    "min_obs_started",
    "duration_minutes",
    "effort_distance_km",
    "number_observers",
    "protocol_type",
    "expertise",
    "julian_date"
  )

  # format for unmarked
  occ_wide <- format_unmarked_occu(occ,
    site_id = "site",
    response = "pres_abs",
    site_covs = c(
      "locality_id", "lc_01.y", "lc_02.y", "lc_05.y",
      "lc_04.y", "lc_09.y", "lc_07.y", "lc_03.y", "bio15a.y", "bio4a.y"
    ),
    obs_covs = obs_covs
  )

  # Convert this dataframe of observations into an unmarked object to start fitting occupancy models
  occ_um <- formatWide(occ_wide, type = "unmarkedFrameOccu")

  model_lc_clim <- occu(~ min_obs_started +
    julian_date +
    duration_minutes +
    effort_distance_km +
    number_observers +
    protocol_type +
    expertise ~ lc_01.y + lc_02.y + lc_05.y +
    lc_04.y + lc_09.y + lc_07.y + lc_03.y + bio15a.y + bio4a.y, data = occ_um)

  occ_gof <- mb.gof.test(model_lc_clim, nsim = 5000, plot.hist = FALSE)

  p.value <- occ_gof$p.value
  c.hat <- occ_gof$c.hat.est
  scientific_name <- unique(data$scientific_name)

  a <- data.frame(scientific_name, p.value, c.hat)

  goodness_of_fit <- rbind(a, goodness_of_fit)

  setTxtProgressBar(pb, i)
}
close(pb)

write.csv(goodness_of_fit, "data/results/05_goodness-of-fit-2.5km.csv")
```

10 Visualizing Occupancy Predictor Effects

In this section, we will visualize the cumulative AIC weights and the magnitude and direction of species-specific probability of occupancy.

To get cumulative AIC weights, we first obtained a measure of relative importance of climatic and landscape predictors by calculating cumulative variable importance scores. These scores were calculated by obtaining the sum of model weights (AIC weights) across all models (including the top models) for each predictor across all species. We then calculated the mean cumulative variable importance score and a standard deviation for each predictor (Burnham and Anderson 2002).

10.1 Prepare libraries

```r
# to load data
library(readxl)
```

```r
# to handle data
library(dplyr)
library(readr)
library(forcats)
library(tidyr)
library(purrr)
library(stringr)
# library(data.table)

# to wrangle models
source("R/fun_model_estimate_collection.r")
source("R/fun_make_resp_data.r")

# nice tables
library(knitr)
library(kableExtra)

# plotting
library(ggplot2)
library(patchwork)
source("R/fun_plot_interaction.r")
```

10.2 Load species list

```r
# list of species
species <- read_csv("data/post_analysis_species_list.csv")
list_of_species <- as.character(species$scientific_name)
```

10.3 Show AIC weight importance

10.3.1 Read in AIC weight data

```r
# which files to read
file_names <- c("data/results/lc-clim-imp.xlsx")

# read in sheets by species
model_imp <- map(file_names, function(f) {
  md_list <- map(list_of_species, function(sn) {

    # some sheets are not found

    tryCatch(
      {
        readxl::read_excel(f, sheet = sn) %>%
          `colnames<-`(c("predictor", "AIC_weight")) %>%
          filter(str_detect(predictor, "psi")) %>%
          mutate(
            predictor = stringr::str_extract(predictor,
              pattern = stringr::regex("\\((.*?)\\)")
            ),
            predictor = stringr::str_replace_all(predictor, "[//(//)]", ""),
            predictor = stringr::str_remove(predictor, "\\.y")
          )
      },
      error = function(e) {
```

```
    message(as.character(e))
   }
  )
 })
 names(md_list) <- list_of_species

 return(md_list)
})
```

10.3.2 Prepare cumulative AIC weight data

```
# assign scale - minimum spatial scale at which the analysis was carried out to account for observer effort
names(model_imp) <- c("2.5km")
model_imp <- imap(model_imp, function(.x, .y) {
  .x <- bind_rows(.x)
  .x$scale <- .y
  return(.x)
})

# bind rows
model_imp <- map(model_imp, bind_rows) %>%
  bind_rows()

# convert to numeric
model_imp$AIC_weight <- as.numeric(model_imp$AIC_weight)
model_imp$scale <- as.factor(model_imp$scale)
levels(model_imp$scale) <- c("2.5km")

# Let's get a summary of cumulative variable importance
model_imp <- group_by(model_imp, predictor) %>%
  summarise(
    mean_AIC = mean(AIC_weight),
    sd_AIC = sd(AIC_weight),
    min_AIC = min(AIC_weight),
    max_AIC = max(AIC_weight),
    med_AIC = median(AIC_weight)
  )

# write to file
write_csv(model_imp,
  file = "data/results/cumulative_AIC_weights.csv"
)
```

Read data back in.

```
# read data and make factor
model_imp <- read_csv("data/results/cumulative_AIC_weights.csv")
model_imp$predictor <- as_factor(model_imp$predictor)
```

```
# make nice names
predictor_name <- tibble(
  predictor = levels(model_imp$predictor),
  pred_name = c(
    "Precipitation Seasonality (CV)",
    "Temperature Seasonality (CV)",
    "% Evergreen Forest", "% Deciduous Forest",
    "% Mixed/Degraded Forest", "% Agriculture/Settlements",
    "% Grassland", "% Plantations", "% Water Bodies"
  )
)

# rename predictor
model_imp <- left_join(model_imp, predictor_name)
```

Prepare figure for cumulative AIC weight. Figure code is hidden in versions rendered as HTML and PDF.

```
fig_aic <-
  ggplot(model_imp) +
  geom_pointrange(aes(
    x = reorder(predictor, mean_AIC),
    y = mean_AIC,
    ymin = mean_AIC - sd_AIC,
    ymax = mean_AIC + sd_AIC
  )) +
  geom_text(aes(
    x = predictor,
    y = 1.2,
    label = pred_name
  ),
  size = 3,
  angle = 0,
  hjust = 1,
  vjust = 2
  ) +
  # scale_y_continuous(breaks = seq(45, 75, 10))+
  scale_x_discrete(labels = NULL) +
  # scale_color_brewer(palette = "RdBu", values = c(0.5, 1))+
  coord_flip(
    ylim = c(0, 1.25)
    # ylim = c(45, 75)
  ) +
  theme_test() +
  theme(legend.position = "none") +
  labs(
    x = "Predictor",
    y = "Cumulative AIC weight"
  )

ggsave(fig_aic,
  filename = "figs/fig_aic_weight.png",
  device = png(),
  dpi = 300,
  width = 79, height = 120, units = "mm"
)
```

10.4 Prepare model coefficient data

For each species, we examined those models which had ΔAICc < 2, as these top models were considered to explain a large proportion of the association between the species-specific probability of occupancy and environmental drivers (Burnham, Anderson, and Huyvaert 2011; Elsen et al. 2017). Using these restricted model sets for each species; we created a model-averaged coefficient estimate for each predictor and assessed its direction and significance (Bartoń

2020). We considered a predictor to be significantly associated with occupancy if the range of the 95% confidence interval around the model-averaged coefficient did not contain zero.

```r
file_read <- c("data/results/lc-clim-modelEst.xlsx")

# read data as list column
model_est <- map(file_read, function(fr) {
  md_list <- map(list_of_species, function(sn) {
    readxl::read_excel(fr, sheet = sn)
  })
  names(md_list) <- list_of_species
  return(md_list)
})

# prepare model data
scales <- c("2.5km")
model_data <- tibble(
  scale = scales,
  scientific_name = list_of_species
) %>%
  arrange(desc(scale))

# rename model data components and separate predictors
names <- c(
  "predictor", "coefficient", "se", "ci_lower",
  "ci_higher", "z_value", "p_value"
)

# get data for plotting:
model_est <- map(model_est, function(l) {
  map(l, function(df) {
    colnames(df) <- names
    df <- separate_interaction_terms(df)
    df <- make_response_data(df)
    return(df)
  })
})

# add names and scales
model_est <- map(model_est, function(l) {
  imap(l, function(.x, .y) {
    mutate(.x, scientific_name = .y)
  })
})

# add names to model estimates
names(model_est) <- scales
model_est <- imap(model_est, function(.x, .y) {
  bind_rows(.x) %>%
    mutate(scale = .y)
})

# remove modulators
model_est <- bind_rows(model_est) %>%
  select(-matches("modulator"))

# join data to species name
model_data <- model_data %>%
  left_join(model_est)

# Keep only those predictors whose p-values are significant:
model_data <- model_data %>%
  filter(p_value < 0.05)
```

369 Export predictor effects.

```r
# get predictor effect data
data_predictor_effect <- distinct(
  model_data,
  scientific_name,
  se,
  predictor, coefficient
)

# write to file
write_csv(data_predictor_effect,
  file = "data/results/data_predictor_effect.csv"
)
```

370 Export model data.

```r
model_data_to_file <- model_data %>%
  select(
    predictor, data,
    scientific_name, scale
  ) %>%
  unnest(cols = "data")

# remove .y
model_data_to_file <- model_data_to_file %>%
  mutate(predictor = str_remove(predictor, "\\.y"))

write_csv(
  model_data_to_file,
  "data/results/data_occupancy_predictors.csv"
)
```

371 Read in data after clearing R session.

```r
# read from file
model_data <- read_csv("data/results/results-predictors-species-traits.csv")
```

372 Fix predictor name.

```r
# remove .y from predictors
model_data <- model_data %>%
  mutate_at(.vars = c("predictor"), .funs = function(x) {
    stringr::str_remove(x, ".y")
  })
```

10.5 Get predictor effects

```r
# is the coeff positive? how many positive per scale per predictor per axis of split?
# now splitting by habitat --- forest or open country
data_predictor <- mutate(model_data,
  direction = coefficient > 0
) %>%
  rename(habitat = "Open-country/Forest") %>%
  count(
    predictor,
    habitat,
    direction
  ) %>%
  mutate(mag = n * (if_else(direction, 1, -1)))

# wrangle data to get nice bars
data_predictor <- data_predictor %>%
  select(-n) %>%
  drop_na(direction) %>%
  mutate(direction = ifelse(direction, "positive", "negative")) %>%
  pivot_wider(values_from = "mag", names_from = "direction") %>%
  mutate_at(
    vars(positive, negative),
    ~ if_else(is.na(.), 0, .)
  )

data_predictor_long <- data_predictor %>%
  pivot_longer(
    cols = c("negative", "positive"),
    names_to = "effect",
    values_to = "magnitude"
  )

# write
write_csv(data_predictor_long,
  path = "data/results/data_predictor_direction_nSpecies.csv"
)
```

Prepare data to determine the direction (positive or negative) of the effect of each predictor. How many species are affected in either direction?

```r
# join with predictor names and relative AIC
data_predictor_long <- left_join(data_predictor_long, model_imp)
```

Prepare figure of the number of species affected in each direction. Figure code is hidden in versions rendered as HTML and PDF.

10.6 Main Text Figure 4

Figure code is hidden in versions rendered as HTML and PDF.

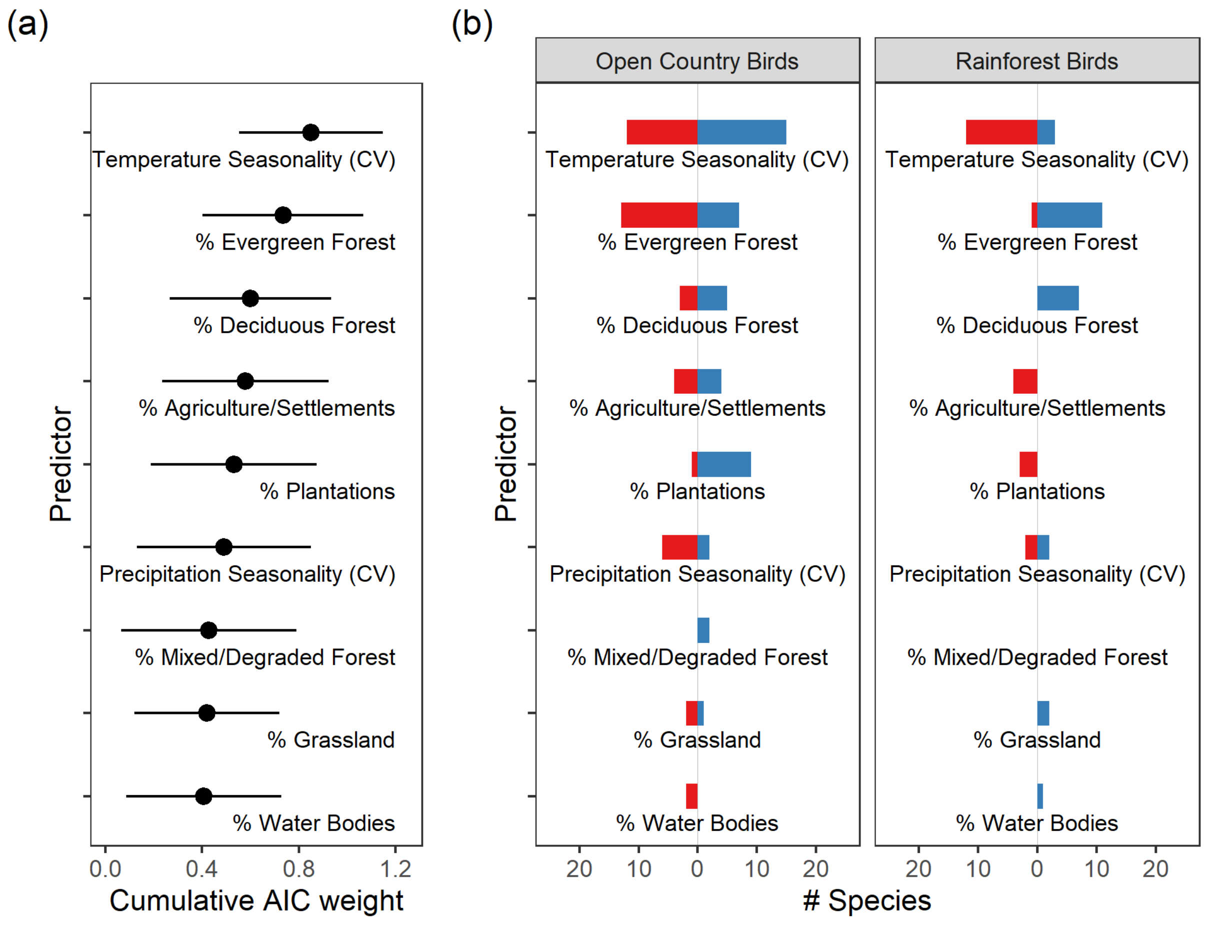

Figure 4: (a) Cumulative AIC weights suggest that climatic predictors have higher relative importance when compared to landscape predictors. (b) The direction of association between species-specific probability of occupancy and climatic and landscape is shown here. While climatic predictors were both positively and negatively associated with the probability of occupancy for a number of species, human-associated land cover types were largely negatively associated with species-specific probability of occupancy.

11 References

Bartoń, Kamil. 2020. *MuMIn: Multi-Model Inference*. Manual.

Burnham, Kenneth P., and David R. Anderson. 2002. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Second. New York: Springer-Verlag. https://doi.org/10.1007/b97636.

Burnham, Kenneth P., David R. Anderson, and Kathryn P. Huyvaert. 2011. "AIC Model Selection and Multimodel Inference in Behavioral Ecology: Some Background, Observations, and Comparisons." *Behavioral Ecology and Sociobiology* 65 (1): 23–35. https://doi.org/10.1007/s00265-010-1029-6.

Elsen, Paul R., Morgan W. Tingley, Ramnarayan Kalyanaraman, Krishnamurthy Ramesh, and David S. Wilcove. 2017. "The Role of Competition, Ecotones, and Temperature in the Elevational Distribution of Himalayan Birds." *Ecology* 98 (2): 337–48. https://doi.org/10.1002/ecy.1669.

Fiske, Ian, and Richard Chandler. 2011. "**Unmarked** : An *R* Package for Fitting Hierarchical Models of Wildlife Occurrence and Abundance." *Journal of Statistical Software* 43 (10). https://doi.org/10.18637/jss.v043.i10.

J., Praveen. 2017. "On the Geo-Precision of Data for Modelling Home Range of a Species A Commentary on Ramesh et Al. (2017)." *Biological Conservation* 213 (September): 245–46. https://doi.org/10.1016/j.biocon.2017.07.017.

Johnston, A, Wm Hochachka, Me Strimas-Mackey, V Ruiz Gutierrez, Oj Robinson, Et Miller, T Auer, St Kelling, and D Fink. 2019. "Analytical Guidelines to Increase the Value of Citizen Science Data: Using eBird Data to Estimate Species Occurrence." Preprint. Ecology. https://doi.org/10.1101/574392.

Johnston, Alison, Daniel Fink, Wesley M. Hochachka, and Steve Kelling. 2018. "Estimates of Observer Expertise Improve Species Distributions from Citizen Science Data." Edited by Nick Isaac. *Methods in Ecology and Evolution* 9 (1): 88–97. https://doi.org/10.1111/2041-210X.12838.

Kelling, Steve, Alison Johnston, Wesley M. Hochachka, Marshall Iliff, Daniel Fink, Jeff Gerbracht, Carl Lagoze, et al. 2015. "Can Observation Skills of Citizen Scientists Be Estimated Using Species Accumulation Curves?" Edited by Stefano Goffredo. *PLOS ONE* 10 (10): e0139600. https://doi.org/10.1371/journal.pone.0139600.

MacKenzie, Darryl I., James D. Nichols, Gideon B. Lachman, Sam Droege, J. Andrew Royle, and Catherine A. Langtimm. 2002. "Estimating Site Occupancy Rates When Detection Probabilities Are Less Than One." *Ecology* 83 (8): 2248–55. https://doi.org/10.1890/0012-9658(2002)083%5B2248:ESORWD%5D2.0.CO;2.

OpenStreetMap contributors. 2019. "Planet Dump Retrieved from https://planet.osm.org."

Sullivan, Brian L., Jocelyn L. Aycrigg, Jessie H. Barry, Rick E. Bonney, Nicholas Bruns, Caren B. Cooper, Theo Damoulas, et al. 2014. "The eBird Enterprise: An Integrated Approach to Development and Application of Citizen Science." *Biological Conservation* 169 (January): 31–40. https://doi.org/10.1016/j.biocon.2013.11.003.

van Strien, Arco J., Chris A. M. van Swaay, and Tim Termaat. 2013. "Opportunistic Citizen Science Data of Animal Species Produce Reliable Estimates of Distribution Trends If Analysed with Occupancy Models." Edited by Vincent Devictor. *Journal of Applied Ecology* 50 (6): 1450–8. https://doi.org/10.1111/1365-2664.12158.