Appendix A1 - Code used for Modelling

Contents

Introduction **2**

Setup Packages **2**

Version control **4**

Occurence Data **5**

Setup Book Data, correct species names and setup species keys for gbif 5

Create multipolygon for all countries for GBIF occ search 8

Create a buffer around PK border and use for masking 9

Downlaod occurences from Gbif 10

Cleaning gbif Data 12

Environmental Data **18**

Main Functions for modelling - Data - Modelling - Evaluation **21**

Function for data subset and train-test data creation for one species 21

Basic modelling functions 23

Ensemble Modelling 29

Function to create prediction raster 31

Basic Function for Evaluation 32

Function for Evaluation for all models over an array of species 33

Processing: Data - Modelling - Evaluation **36**

Processing evaluation data for PA1 **38**

Processing evaluation data for PA10 **50**

Plotting and Analyzing evaluation **62**

Heattable for ranks 62

Calculate Mean Model Ranks 65

Compare Performance for multiple threshholds 67

. 69

Introduction

This is the code used for the Bachelor-Thesis "Modelling Patterns of Butterfly Species Richness in Pakistan: Stacking species distribution models using an ensemble of tuned models". While it was attempted to keep it as easy as possible to reproduce there are some limitations for that:

- this code does not create a folder structure but it relies on one
  - this means that folders need to be created before running a code chunk
- there might be slight incontinuities in naming inside the code
  - the reason for that is that the code was used over the course of multiple months
- and many different versions were used

Setup Packages

```
# Packages ---------------------------------------------


# package by valavi et al. used for GAM
# remotes::install_github('rvalavi/myspatial')


# create operatorby negating 'in' operator
`%not_in%` <- Negate(`%in%`)

# Load Packages
packages <- c("here", "glue", "tidyverse", "data.table", "terra",
    "sf", "dplyr", "raster", "raptr", "nngeo", "rnaturalearth",
    "geodata", "rgbif", "CoordinateCleaner", "countrycode", "clhs",
    "mgcv", "glmnet", "randomForest", "dismo", "myspatial", "leaflet",
    "ggplot2", "ggrepel", "ggforce", "snowfall")


# Function to install and load packages
# ---------------------------------
```

```r
install_and_load_packages <- function(packages) {
    # Loop through each package in the vector
    for (pkg in packages) {
        # Check if the package is installed
        if (!requireNamespace(pkg, quietly = TRUE)) {
            # If not installed, install the package
            install.packages(pkg)
        }
        # Load the package into the R session
        library(pkg, character.only = TRUE)
    }
}


# load packages
install_and_load_packages(packages)
```

```
## Warning: package 'here' was built under R version 4.2.3

## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.2

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.2

## Warning: package 'dplyr' was built under R version 4.2.3

## Warning: package 'stringr' was built under R version 4.2.2

## Warning: package 'forcats' was built under R version 4.2.2

## Warning: package 'lubridate' was built under R version 4.2.3

## Warning: package 'data.table' was built under R version 4.2.3

## Warning: package 'terra' was built under R version 4.2.3

## Warning: package 'sf' was built under R version 4.2.3

## Warning: package 'raster' was built under R version 4.2.3

## Warning: package 'sp' was built under R version 4.2.2
```

```
## Warning: package 'raptr' was built under R version 4.2.3

## Warning: package 'nngeo' was built under R version 4.2.3

## Warning: package 'rnaturalearth' was built under R version 4.2.2

## Warning: package 'geodata' was built under R version 4.2.3

## Warning: package 'rgbif' was built under R version 4.2.3

## Warning: package 'CoordinateCleaner' was built under R version 4.2.3

## Warning: package 'countrycode' was built under R version 4.2.3

## Warning: package 'clhs' was built under R version 4.2.3

## Warning: package 'mgcv' was built under R version 4.2.3

## Warning: package 'glmnet' was built under R version 4.2.3

## Warning: package 'Matrix' was built under R version 4.2.3

## Warning: package 'randomForest' was built under R version 4.2.3

## Warning: package 'dismo' was built under R version 4.2.2

## Warning: package 'leaflet' was built under R version 4.2.3

## Warning: package 'ggrepel' was built under R version 4.2.3

## Warning: package 'ggforce' was built under R version 4.2.3

## Warning: package 'snowfall' was built under R version 4.2.1
```

Version control

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22621)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.utf8
## [2] LC_CTYPE=English_United Kingdom.utf8
```

```
## [3] LC_MONETARY=English_United Kingdom.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] snowfall_1.84-6.2       snow_0.4-4              ggforce_0.4.1
##  [4] ggrepel_0.9.3           leaflet_2.1.2           myspatial_0.1.4
##  [7] dismo_1.3-9             randomForest_4.7-1.1    glmnet_4.1-7
## [10] Matrix_1.5-4            mgcv_1.8-42             nlme_3.1-157
## [13] clhs_0.9.0              countrycode_1.5.0       CoordinateCleaner_2.0-20
## [16] rgbif_3.7.7             geodata_0.5-8           rnaturalearth_0.3.2
## [19] nngeo_0.4.7             raptr_1.0.0             raster_3.6-20
## [22] sp_1.6-0                sf_1.0-12               terra_1.7-29
## [25] data.table_1.14.8       lubridate_1.9.2         forcats_1.0.0
## [28] stringr_1.5.0           dplyr_1.1.2             purrr_1.0.1
## [31] readr_2.1.4             tidyr_1.3.0             tibble_3.2.1
## [34] ggplot2_3.4.2           tidyverse_2.0.0         glue_1.6.2
## [37] here_1.0.1
##
## loaded via a namespace (and not attached):
##  [1] oai_0.4.0          httr_1.4.5         rprojroot_2.0.3    tools_4.2.0
##  [5] utf8_1.2.3         rgdal_1.6-6        R6_2.5.1           KernSmooth_2.23-20
##  [9] rgeos_0.6-2        DBI_1.1.3          lazyeval_0.2.2     colorspace_2.1-0
## [13] withr_2.5.0        tidyselect_1.2.0   compiler_4.2.0     cli_3.6.1
## [17] formatR_1.14       xml2_1.3.3         scales_1.2.1       classInt_0.4-9
## [21] proxy_0.4-27       digest_0.6.31      rmarkdown_2.21     pkgconfig_2.0.3
## [25] htmltools_0.5.5    fastmap_1.1.1      htmlwidgets_1.6.2  rlang_1.1.0
## [29] rstudioapi_0.14    farver_2.1.1       shape_1.4.6        generics_0.1.3
## [33] jsonlite_1.8.4     crosstalk_1.2.0    magrittr_2.0.3     geosphere_1.5-18
## [37] Rcpp_1.0.10        munsell_0.5.0      fansi_1.0.4        lifecycle_1.0.3
## [41] stringi_1.7.12     whisker_0.4.1      yaml_2.3.7         MASS_7.3-56
## [45] plyr_1.8.8         grid_4.2.0         parallel_4.2.0     lattice_0.20-45
## [49] splines_4.2.0      hms_1.1.3          knitr_1.42         pillar_1.9.0
## [53] reshape2_1.4.4     codetools_0.2-18   evaluate_0.20      tweenr_2.0.2
## [57] vctrs_0.6.2        tzdb_0.3.0         foreach_1.5.2      polyclip_1.10-4
## [61] gtable_0.3.3       xfun_0.39          e1071_1.7-13       class_7.3-20
## [65] survival_3.3-1     iterators_1.0.14   units_0.8-1        cluster_2.1.4
## [69] timechange_0.2.0
```

Occurence Data

Setup Book Data, correct species names and setup species keys for gbif

```
# Read book data from Tshikolovets
# ------------------------------------------------------------

# Read data
occurence_data <- read.csv(here::here("data/raw_data/pakistan_ladakh.csv"))
```

```r
# correct names
names(occurence_data) <- c("species", "x", "y")

# add column to indicate data source
occurence_data$retrieved <- "BOOK"

# add column to show country of record
occurence_data$country <- "Pakistan"

# GBIF species names
# ------------------------------------------------------

# all unique species names
all_species_unique <- unique(occurence_data$species)

# for all species search for equivalant species in gbif
# taxonomy and create a data frame with meta data
gbif_name_checklist <- rgbif::name_backbone_checklist(all_species_unique)

#
gbif_name_checklist_filtered <- gbif_name_checklist %>%  gbif_name_checklist_filtered
gbif_name_checklist_filtered <- gbif_name_checklist %>%  <-
gbif_name_checklist_filtered <- gbif_name_checklist %>%  gbif_name_checklist
gbif_name_checklist_filtered <- gbif_name_checklist %>%  %>%
# filter for all entries for which a species was found
# (excluding entries for which genus or none taxon was
# found)
filter(rank == "SPECIES")

# limit to relevant columns
gbif_name_checklist_screening <- gbif_name_checklist[, c("species",
    "verbatim_name", "scientificName", "rank", "status", "confidence",
    "matchType")]


# screen names
# ------------------------------------------------

# setup dataframe for screening
gbif_name_screening_fuzzy <- gbif_name_checklist_screening[gbif_name_checklist_screening$matchType ==
    "FUZZY", ]

# species which were found to be incorrect
wrong_fuzzy <- c("Coladenia_indrani", "Hyponephele_pulchra",
    "Lasiommata_maerula", "Polyommatus_chrysopis", "Ypthima_sakra")


# vector with species names to exclude from gbif search
# combining screening results with gbif taxons not on
# species level
exclude_species <- c(wrong_fuzzy, gbif_name_checklist$verbatim_name[gbif_name_checklist$rank !=
    "SPECIES"])
```

```r
# exclude species
# -------------------------------------------------------



# setup dataframe containing all the taxon keys, species
# names used for search and resulting scientific names
gbif_keys_names <- as.data.frame(dplyr::filter(gbif_name_checklist_filtered,
    verbatim_name %not_in% exclude_species))[, c("usageKey",
    "verbatim_name", "scientificName", "species")]



# correct book and gbif species names
# ------------------------------------

# sepearte scientific names by ' '
gbif_scientific <- stringr::str_split(gbif_keys_names$scientificName,
    " ")

# loop over all species used for gbif search to use the
# scientific name to correct the book data name and for the
# gbif search
for (i in 1:length(gbif_scientific)) {

    # safe list elements in variables and combine them to
    # one string scientific name genus
    a <- gbif_scientific[[i]][1]

    # seperator
    b <- "_"

    # scientific name species epithet
    c <- gbif_scientific[[i]][2]

    # combine strings
    abc <- str_c(a, b, c)

    # correct book data
    occurence_data$species[occurence_data$species == gbif_keys_names$verbatim_name[i]] <- abc

    # write correct names in gbif_keys_names
    gbif_keys_names$original_species[i] <- abc
}


# write data as csv s
write_csv(gbif_keys_names, here::here("data/processed_data/gbif_keys_names_extent.csv"))
```

Create multipolygon for all countries for GBIF occ search

```
# set up vector containing country names
countries = c("india", "pakistan", "afghanistan", "tajikistan",
    "iran", "china", "nepal", "uzbekistan", "kyrgyzstan", "turkmenistan",
    "kazakhstan")

# Define the target CRS
target_crs <- sf::st_crs("EPSG:4326")

# setup sf data as an sfc (spatial feature collection)
single_sf <- sf::st_sfc()

# loop to get necessary country borders and merge them
# together
for (i in countries) {

    # download country data through package 'rnaturalearth'
    border <- rnaturalearth::ne_countries(scale = 50, type = "countries",
        country = i, returnclass = "sf")

    # use epsg 4326 as crs for the sf object
    border_4326 <- sf::st_transform(border, target_crs)

    # correct invalid geometries
    border_fixed <- sf::st_make_valid(border_4326)

    # combine all borders into an sfc collection
    if (i == countries[1]) {
        single_sf <- border_fixed
    } else {
        single_sf <- dplyr::bind_rows(list(single_sf, border_fixed))
    }
}

# merge sfc collection to a single multipolygon (all
# countries as one multipolygon)
union_poly <- st_union(single_sf)

# fill holes in multipolygon
union_poly_filled <- nngeo::st_remove_holes(union_poly)
rm(single_sf, union_poly)

# plot data
plot(union_poly_filled)

# write union_poly_filled into shp file
write_sf(union_poly_filled, here::here("data/processed_data/shapes/union_poly.shp"))
```

Create a buffer around PK border and use for masking

```
# shape borders
shape_pakistan <- ne_countries(scale = 50, type = "countries",
    country = "pakistan", returnclass = "sf")

# Transform the polygon to the target CRS
pakistan_4326 <- st_transform(shape_pakistan, target_crs)
rm(shape_pakistan)

# Fix the polygon geometry
pakistan_fixed <- st_make_valid(pakistan_4326)
rm(pakistan_4326)

# create buffer around pakistan border of 1000 km
pbuf_basic = st_buffer(pakistan_fixed, dist = 1e+06)

# for later use we need to reduce the number of points in
# the polygon so we only use every second point

# safe polygon as wkt
pbuf_wkt <- sf::st_as_text(pbuf_basic$geometry[[1]])

# seperate points building the polygon
pbuf_points <- stringr::str_split_1(pbuf_wkt, pattern = ", ")

# select every second point
pbuf_points_reduced <- pbuf_points[seq(1, length(pbuf_points),
    2)]

# merge back together to form polygon with half the number
# of points used later for gbif occurence download
pbuf_wkt_reduced <- stringr::str_flatten(pbuf_points_reduced,
    collapse = ", ")


# cropping the pakistan buffer by union_poly_filled for
# cropping occurrence records and environmental data later
# on first safe as pbuf as sfc
pbuf_sfc <- sf::st_as_sfc(pbuf_wkt_reduced)

# then as sf
pbuf_sf <- sf::st_sf(geometry = pbuf_sfc, crs = target_crs)

# intersect pbuf_ with countries wanted to be included
pbuf_sf_final <- sf::st_intersection(pbuf_sf, union_poly_filled)

# Plot
plot(pbuf_sf_final)
```

Downlaod occurences from Gbif

```
# function to download occurrences from gbif  ------------------------------

# key_list should contain the usage key for all species as used for gbif
# taxon_list should include all species names (with or without space)
# extent_wkt is the geometry limiting the search area given as a wkt (string)
occurence_download_orig <- function(key_list, taxon_list, extent_wkt) {
  # set up data frames
  gbif_data <- data.frame()
  taxons_with_synonym <- data.frame()

  # loop over taxon_list
  for (i in 1:length(taxon_list)) {
    # catch errors
    tryCatch(
      {
        # safe current taxon name to variable
        taxon_name_char <- taxon_list[i]

        # print out current taxon name
        print(glue::glue("{i}: searching for {taxon_name_char}"))

        # safe current taxon key to variable
        taxon_key <- key_list[i]

        # check for number of available occurrences
        occ_count <- rgbif::occ_count(
          geometry = extent_wkt, # give extent for search
          taxonKey = taxon_key, # give taxon key
          occurrenceStatus = "PRESENT", # look for presence occurrences
          hasCoordinate = T # needs to have coordinates
        )

        # if more than 0 occurrences were found
        if (occ_count > 0 & !is.null(taxon_key) & taxon_data$matchType == "EXACT") {
          # search for occurrences (and safe data to species_occurrences)
          species_occurrences <- rgbif::occ_search(
            geometry = extent_wkt, # give extent for search
            taxonKey = taxon_key, # give taxon key
            hasCoordinate = T # needs to have coordinates
          )

          # for the case that no of the found occurrences has data for "year" (was source of error))
          if (c("year") %in% names(species_occurrences$data)) {
            occ_df <- data.frame(
              # selected columns from the data
              name_accepted = if ("species" %in% names(species_occurrences$data)) {
                species_occurrences$data$species
              } else {
                "None"
              },
              name_original = rep(taxon_name_char, nrow(species_occurrences$data)), # search name
```

```
            # accepted scientific name
            accepted_science_name = species_occurrences$data$acceptedScientificName,
            original_science_name = species_occurrences$data$scientificName, # scientific name
            accepted_taxon_key = species_occurrences$data$acceptedTaxonKey,
            original_taxon_key = species_occurrences$data$taxonKey, # key
            x = species_occurrences$data$decimalLongitude, # Longitude
            y = species_occurrences$data$decimalLatitude, # Latitude
            year = species_occurrences$data$year, # year of record
            country = species_occurrences$data$country, # country of record
            basis_of_record = species_occurrences$data$basisOfRecord, # type of observation
            geodetic_datum = species_occurrences$data$geodeticDatum, # geodetic Datum
            # data set key (useful to find citation and more info)
            dataset_key = species_occurrences$data$datasetKey,
            status = species_occurrences$data$occurrenceStatus # Present or Absent
          )

          # bind data for current species to data
          gbif_data <- rbind(gbif_data, occ_df)

          # if occurrences found and column year exists print:
          print(glue::glue("{i}: {occ_count} occurrences were found for {
                           taxon_name_char}  as
                           {species_occurrences$data$species[1]}"))

        } else { # if colum year not found print:
          print(glue("{i}: No year of record for occurrences"))
        }
      } else { # if no occurrences found print:
        print(glue::glue("{i}: No occurrences found for {taxon_name_char}"))
      }

      # if last species reached:
      if (i == length(taxon_list)) {
        return(gbif_data)
      }
    }, # end of trycatch code Input

    # error output:
    error = function(e) {
      if (!exists("error_list")) {
        error_list <- list()
      }
      error_message <- message(glue::glue({
        "An error occurred for species {taxon_name_char} ({i}): "
      }), conditionMessage(e))
      error_list <- append(error_list, error_message)
    }
  )
 } # end of for loop
} # end of function
```

```r
# download data
extent_gbif_data <- occurence_download_orig(
  gbif_keys_names$usageKey,
  gbif_keys_names$species,
  pbuf_wkt_reduced
)


# gbif data as sf
extent_gbif_data_sf <- sf::st_as_sf(extent_gbif_data, coords = c("x", "y"), crs = target_crs)

# intersect to get only occurences also falling in the counties included
extent_gbif_data_intersect <- sf::st_intersection(extent_gbif_data_sf, union_poly_filled)

# split geometry into x and y coordinates column
extent_gbif_data_download <- st_drop_geometry(extent_gbif_data_intersect) %>%
  mutate(
    x = unlist(map(extent_gbif_data_intersect$geometry, 1)),
    y = unlist(map(extent_gbif_data_intersect$geometry, 2))
  )

# safe data
write_csv(extent_gbif_data_download, here::here("data/processed_data/extent_gbif_data_download.csv"))
```

Cleaning gbif Data

```r
# set up data ------------------------------------------

# create operator
`%not_in%` <- Negate(`%in%`)

# gbif data
gbif_data <- read.csv(here::here("data/processed_data/extent_gbif_data_download.csv"))
gbif_data$retrieved <- "GBIF"
colnames(gbif_data)[colnames(gbif_data) == "name_original"] <- "species"




# Filter gbif data by year -----------------------------------------------

# breakpoints
br <- c(1700, 1800, 1900, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2020, 2023)

# range
ranges <- paste(head(br, -1), br[-1], sep = " - ")

# frequency
freq <- hist(gbif_data$year, breaks = br, include.lowest = TRUE, plot = FALSE)
```

```r
# frequency table
data.frame(range = ranges, frequency = freq$counts)

gbif_data <- dplyr::filter(gbif_data, year >= 1850)


# Combine book and gbif data ------------------------------------------------------------

# check meta data
gbif_data[gbif_data$geodetic_datum != "WGS84" |
  gbif_data$status != "PRESENT" |
  (gbif_data$basis_of_record != "MATERIAL_SAMPLE" &
    gbif_data$basis_of_record != "HUMAN_OBSERVATION" &
    gbif_data$basis_of_record != "PRESERVED_SPECIMEN" &
    gbif_data$basis_of_record != "OCCURRENCE"), ]

# filter gbif colums
gbif_data_link <- gbif_data[, names(gbif_data) %in% names(occurence_data)]


# combine book and gbif data
full_data <- rbind(occurence_data, gbif_data_link)

unique(full_data$species) # result should be 429



## filter and control data ---------------------------------------------------

# add 3 letter country code for coordinate cleaner package
full_data$country_code <- countrycode::countrycode(full_data$country,
  origin = "country.name",
  destination = "iso3c"
)

# create flags
flags <- CoordinateCleaner::clean_coordinates(
  x = full_data,
  lon = "x",
  lat = "y",
  countries = "country_code",
  species = "species",
  tests = c("countries", "equal", "zeros", "seas", "duplicates")
) # most test are on by default


# change country name of Iran
flags$country[flags$country == "Iran (Islamic Republic of)"] <- "Iran"

# head flags
# if a flag value is TRUE, it means it is not flagged for that category
head(flags)
```

```r
# summary flags
summary(flags)


# Cleaning coordinates  -----------------------------------------------------------

## function to screen country flags ------------------------------------------

# function to create maps showing flaged occurences for country not matching coordinates
occ_oob <- function(flags, country = "all_countries", country_sea_both = "both",
                    target_crs = st_crs("EPSG:4326"), x_lim = c(0, 0), y_lim = c(0, 0),
                    extent_poly) {
  require(glue)
  require(ggplot2)
  require(stringr)
  require(sf)

  tryCatch(
    {
      # create worldmap
      wm <- ggplot2::borders("world", colour = "gray50", fill = "gray50")

      # create 4 digit country name
      country_name <- stringr::str_split(country, pattern = " ")[[1]]

      # print name
      print(country_name)

      # flag subset country without ea

      # subset flags data depending on sea_country_both
      # include entries flagged as on the sea
      # (all of them fall into category of not in the right country)
      if (country_sea_both == "both") {
        flag_subset <- flags[flags$.con == FALSE &
          (if (country != "all_countries") {
            flags$country == country
          } else {
            TRUE
          }), ]

        # only include entries flagged as not in the right country but not flagged as on the sea
      }
      if (country_sea_both == "country") {
        flag_subset <- flags[flags$.sea == TRUE &
          flags$.con == FALSE &
          (if (country != "all_countries") {
            flags$country == country
          } else {
            TRUE
          }), ]
      }
```

```r
if (country_sea_both == "sea") {
  flag_subset <- flags[flags$.sea == FALSE &
    (if (country != "all_countries") {
      flags$country == country
    } else {
      TRUE
    }), ]
}

# create sf object of flags
flag_subset_sf <- st_as_sf(flag_subset, coords = c("x", "y"), crs = target_crs)

# which of these flagged points is in none of the neighboring countries to PK

occurence_in_bounds <- st_intersection(flag_subset_sf, extent_poly)
occurence_out_of_bounds <- flag_subset_sf[
  flag_subset_sf$geometry %not_in% occurence_in_bounds$geometry,
]

assign(
  x = glue("occurence_out_of_bounds_{country_name}"),
  value = occurence_out_of_bounds,
  envir = parent.frame()
)

# set up frame for plot
if (x_lim[1] == 0 & x_lim[2] == 0 & y_lim[1] == 0 & y_lim[2] == 0) {
  y_lim <- c(
    st_bbox(occurence_out_of_bounds)["ymin"][[1]] - 10,
    st_bbox(occurence_out_of_bounds)["ymax"][[1]] + 10
  )

  x_lim <- c(
    st_bbox(occurence_out_of_bounds)["xmin"][[1]] - 10,
    st_bbox(occurence_out_of_bounds)["xmax"][[1]] + 10
  )
}

# print number of flags
print(glue("Occurences out of bounds: {nrow(occurence_out_of_bounds)}"))

# create plot
map_country <- ggplot() +
  ggplot2::ggtitle(glue::glue("{country_name} has
                              {nrow(occurence_out_of_bounds)} entries
                              with {country_sea_both} flagged")) +
  wm +


  geom_sf(data = extent_poly, aes(geometry = geometry)) +
  geom_sf(data = occurence_out_of_bounds, aes(geometry = geometry,
                                              colour = rownames(occurence_out_of_bounds)),
          size = 1.2) +
```

```
        scale_color_manual(values = rep(rainbow(26, s = .6, v = .9),
                                        nrow(occurence_out_of_bounds))) +
        theme(legend.position = "none") +
        ggplot2::coord_sf(
          ylim = y_lim,
          xlim = x_lim
        )

      # create names for plots
      name_map <- glue::glue("{country_name}_flag_subset_map")

      # assign plots to name
      assign(name_map,
        map_country,
        envir = parent.frame()
      )

      # print plot
      print(map_country)
    }, # end of tryCatch code input

    # output error message if error occurred
    error = function(e) {
      message(glue::glue("error for {country_name}: "), conditionMessage(e))
    }
  ) # end of tryCatch call
} # end of function




# plot flags
plot(flags, lon = "x", lat = "y")

# create flags sf
flags_sf <- sf::st_as_sf(flags, coords = c("x", "y"), crs = target_crs)

# function that checks if occurences are in the specified area of our extent
# can also be used to screen country and sea flagged entries
# in our case all entries not in extent_shape have already been filtered
occ_oob(
  flags = flags_sf, country = "all_countries", country_sea_both = "both",
  target_crs = target_crs, x_lim = c(0, 0), y_lim = c(0, 0),
  extent_poly = union_poly_filled
)


# get rid of duplicates
occ_data_clean <- dplyr::distinct(full_data,
  x, y, species,
  .keep_all = TRUE
)
```

```r
occ_data_clean$occ <- 1

# check flags again
flags <- CoordinateCleaner::clean_coordinates(
  x = occ_data_clean,
  lon = "x",
  lat = "y",
  countries = "country_code",
  species = "species",
  tests = c("countries", "equal", "zeros", "seas", "duplicates")
)

# summary flags
summary(flags)

# still some country flagged entries left, but all of them are inside extent_shape
# and have been screened.


# citation ----------------------------------------------------------------

# select data used from original gbif data
data_set_keys <- unique(
  gbif_data$dataset_key[gbif_data$x %in% occ_data_clean$x & gbif_data$y %in% occ_data_clean$y]
)

# for loop to run over all data sat keys used in the gbif data to get citation for all data used
for (i in data_set_keys) {
  # citation for dataset_key i
  cite_temp <- rgbif::gbif_citation(x = i)

  # create citation data frame
  cite_temp_df <- data.frame(
    dataset_key = i,
    title = cite_temp$citation$title,
    text = cite_temp$citation$text,
    citation = cite_temp$citation$citation,
    accessed = cite_temp$citation$accessed
  )


  # if data frame doesnt exist yet
  if (!exists("gbif_cite")) {
    gbif_cite <- cite_temp_df

    # else row bind
  } else {
    gbif_cite <- rbind(gbif_cite, cite_temp_df)
  }
} # end of for loop

head(gbif_cite)
```

```r
head(occ_data_clean)

# write data to disk ------------------------------------------------

data.table::fwrite(occ_data_clean, here::here("data/processed_data/final/occ_data_clean.csv"))
data.table::fwrite(gbif_cite, here::here("data/processed_data/final/gbif_cite.csv"))
```

Environmental Data

```r
# Environmental data
# ------------------------------------------------------

# file lists
chelsa_file_list <- list.files(path = here::here("data/raw_data/chelsa"),
    pattern = ".tif", all.files = TRUE, full.names = TRUE)

# Load raster data
chelsa_raw <- sapply(chelsa_file_list, terra::rast)

rm(chelsa_file_list)


# setup correct layer names
bio_names <- stringr::str_replace_all(str = c("Bio 1 Annual Mean Temperature",
    "Bio 10 Mean Temperature of Warmest Quarter", "Bio 11 Mean Temperature of Coldest Quarter",
    "Bio 12 Annual Precipitation", "Bio 13 Precipitation of Wettest Month",
    "Bio 14 Precipitation of Driest Month", "Bio 15 Precipitation Seasonality",
    "Bio 16 Precipitation of Wettest Quarter", "Bio 17 Precipitation of Driest Quarter",
    "Bio 18 Precipitation of Warmest Quarter", "Bio 19 Precipitation of Coldest Quarter",
    "Bio 2 Mean Diurnal Range", "Bio 3 Isothermality", "Bio 4 Temperature Seasonality",
    "Bio 5 Max Temperature of Warmest Month", "Bio 6 Min Temperature of Coldest Month",
    "Bio 7 Temperature Annual Range", "Bio 8 Mean Temperature of Wettest Quarter",
    "Bio 9 Mean Temperature of Driest Quarter", "cmi_max", "cmi_mean",
    "cmi_min", "cmi_range", "ngd5", "pet_penman_max", "pet_penman_mean",
    "pet_penman_min", "pet_penman_range"), pattern = " ", replacement = "_")

# correct layer names
names(chelsa_raw) <- bio_names

rm(bio_names)

# create spatVector from full shape polygon (all countries)
# union_terra <- vect(union_poly_filled)

# Crop to Pakistan Extent
chelsa_crop <- lapply(chelsa_raw, FUN = terra::crop, y = terra::ext(pbuf_sf_final))



rm(chelsa_raw)
```

```r
# Mask to Pakistan extent

chelsa_mask <- lapply(chelsa_crop, FUN = terra::mask, mask = terra::vect(pbuf_sf_final))

rm(chelsa_crop)

# create one rast object with 27 dimensions from 27 layers
# of chelsa mask
chelsa <- terra::rast(chelsa_mask)

rm(chelsa_mask)

# safe data to disk
terra::writeRaster(chelsa, filename = here("data/processed_data/environmental/chelsa_buffed.tif"),
    overwrite = T)

# setup as env data as data frame
chelsa_df <- terra::as.data.frame(chelsa, cells = TRUE)



# Colinearity
# ------------------------------------------------------------



# cell column needs to be removed as well as variable
# Bio_7_Temperature_Annual_Range (problem with data)
chelsa_df_vif <- chelsa_df[, -c(1, 18)]

# load functions from HighstatLib by Zurr et al
# (https://doi.org/10.1111/j.2041-210X.2009.00001.x)
# although it says VIF was analyzed this function also show
# pearson correlation and this was actually used to check
# colinearity
source(here::here("process/scripts/zurr_2010/HighstatLib.r"))

# corvif of all variables
corvif(chelsa_df_vif)


# temperature variable names as in chelsa_df
temp_variables <- c("Bio_1_Annual_Mean_Temperature", "Bio_10_Mean_Temperature_of_Warmest_Quarter",
    "Bio_11_Mean_Temperature_of_Coldest_Quarter", "Bio_2_Mean_Diurnal_Range",
    "Bio_3_Isothermality", "Bio_4_Temperature_Seasonality", "Bio_5_Max_Temperature_of_Warmest_Month",
    "Bio_6_Min_Temperature_of_Coldest_Month", "Bio_8_Mean_Temperature_of_Wettest_Quarter",
    "Bio_9_Mean_Temperature_of_Driest_Quarter", "ngd5", "cmi_mean")

# precipitation variable names as in chelsa_df
precip_variables <- c("Bio_12_Annual_Precipitation", "Bio_13_Precipitation_of_Wettest_Month",
    "Bio_14_Precipitation_of_Driest_Month", "Bio_15_Precipitation_Seasonality",
    "Bio_16_Precipitation_of_Wettest_Quarter", "Bio_17_Precipitation_of_Driest_Quarter",
    "Bio_18_Precipitation_of_Warmest_Quarter", "Bio_19_Precipitation_of_Coldest_Quarter",
```

```
    "cmi_mean")


# temp vif
corvif(chelsa_df_vif[, temp_variables])

# precip vif
corvif(chelsa_df_vif[, precip_variables])
# precip of driest quarter over precip of driest month
# precip of hottest quarter over annual precipitation
# precip of wettest quarter over precip of wettest month




# selection of temperature variables starting with min
# temperature of coldest month and cmi_mean
temp_selec_1 <- c("Bio_6_Min_Temperature_of_Coldest_Month", "Bio_2_Mean_Diurnal_Range",
    "cmi_mean", "Bio_8_Mean_Temperature_of_Wettest_Quarter",
    "Bio_4_Temperature_Seasonality", "Bio_3_Isothermality")
# VIF
corvif(chelsa_df_vif[, temp_selec_1])

# 5 precipitation left
precip_selec_1 <- c("Bio_14_Precipitation_of_Driest_Month", "Bio_18_Precipitation_of_Warmest_Quarter",
    "Bio_19_Precipitation_of_Coldest_Quarter", "cmi_mean", "Bio_13_Precipitation_of_Wettest_Month",
    "Bio_15_Precipitation_Seasonality")
# VIF
corvif(chelsa_df_vif[, precip_selec_final])




# further selection

precip_selec_final <- c("Bio_14_Precipitation_of_Driest_Month",
    "Bio_19_Precipitation_of_Coldest_Quarter", "Bio_18_Precipitation_of_Warmest_Quarter",
    "Bio_13_Precipitation_of_Wettest_Month", "Bio_15_Precipitation_Seasonality")



temp_selec_final <- c("Bio_6_Min_Temperature_of_Coldest_Month",
    "Bio_4_Temperature_Seasonality", "Bio_2_Mean_Diurnal_Range")

# Check correlation
corvif(chelsa_df_vif[, temp_selec_final])
corvif(chelsa_df_vif[, precip_selec_final])


# combine precipitation and temperature variable selection
# to one vector
final_selec <- c(temp_selec_final, precip_selec_final)
```

```r
# check vif for final variables
corvif(chelsa_df_vif[, final_selec])

# subset rast and env data with final selection of
# variables
env_data_rast <- terra::subset(chelsa, final_selec)

env_data_df <- terra::as.data.frame(env_data_rast, cells = TRUE)

# safe env_data_rast to disk
terra::writeRaster(env_data_rast, "data/processed_data/final/env_data_rast_new.tif",
    overwrite = TRUE)

## Conditioned Latin Hypercube Sampling with for 10.000
## background points
clhs_index_20000 <- clhs(env_data_df[, names(env_data_df) !=
    "cell"], size = 20000, progress = FALSE, simple = TRUE)
saveRDS(clhs_index_20000, here::here("data/processed_data/environmental/clhs_20000_new.RData"))
```

Main Functions for modelling - Data - Modelling - Evaluation

Function for data subset and train-test data creation for one species

```r
# Function to subset species
# --------------------------------------------

subset_species <- function(data, species_name, path_env_data_rast,
    env_data_df, clhs_ind) {

    require(dplyr)
    require(sf)
    require(raster)
    require(dismo)

    # presence data
    # ------------------------------------------------------------

    temp <- Sys.time()

    print(glue("Starting subset data of {species_name}"))

    env_data_rast <- terra::rast(path_env_data_rast)

    # Choose species
    data_filtered <- as.data.frame(dplyr::filter(data, species ==
        species_name))

    # Create SF Object with Epsg 4326 as CRS
    data_filtered_sf <- sf::st_as_sf(data_filtered, coords = c("x",
```

```
    "y"), remove = F, crs = sf::st_crs("epsg:4326"))

# extract Env_data
data_extr = terra::extract(env_data_rast, data_filtered_sf)

# add env_data
data_filtered <- cbind(data_filtered, data_extr)
rm(data_extr)

# subset 20 % of the data as test data
test_data_with_meta <- dplyr::slice_sample(data_filtered,
    prop = 0.2)

# select train data by excluding rows with ID that is
# already in test data
train_data_with_meta <- dplyr::filter(data_filtered, ID %not_in%
    test_data_with_meta$ID)

# subset data to only get relevant columns for
# modelling select by number of columns of env_data_df
# + 2 because of cell column
train_data <- train_data_with_meta[, (ncol(train_data_with_meta) -
    ncol(env_data_df) + 2):ncol(train_data_with_meta)]
train_data$occ <- 1

# background data
# ------------------------------------------------------

# extract with clhs as index from environmental
# variable data frame
background_data <- env_data_df[clhs_ind, names(env_data_df) !=
    "cell"]

# get cell numbers of presence data
cells_presence_data <- terra::extract(env_data_rast[[1]],
    vect(data_filtered_sf), cells = TRUE)$cell

# vector specifiying indexes for overlapping points
# (points are in same cell)
overlapping_bg <- which(as.integer(row.names(background_data)) %in%
    cells_presence_data)

# if there is any overlapping points
if (length(overlapping_bg) != 0) {

    # get rid of background data points overlapping
    # with presences
    background_data <- background_data[-overlapping_bg, ]

}

# add occurrence column = 0
background_data$occ <- 0
```

```
    # all cells background and presence data
    cells_all_data <- c(overlapping_bg, as.integer(row.names(background_data)))

    # create 1000 random points inside SpatRaster
    background_test_data <- raptr::randomPoints(env_data_rast[[1]],
        2000)

    # background_test_data cells in spatRaster
    cells_bg_test <- terra::extract(env_data_rast[[1]], vect(background_test_data),
        cells = TRUE)$cell

    # vector specifiying indexes for overlapping points
    # (points are in same cell)
    overlapping_bg_test <- which(cells_bg_test %in% cells_all_data)

    # if there is any overlapping points
    if (length(overlapping_bg_test) != 0) {

        # get rid of background data points overlapping
        # with presences
        background_test_data <- background_test_data[-overlapping_bg_test,
            ]
    }

    # combine presence and background data
    pb_data <- rbind(train_data, background_data)

    # combine to a list: pb_data (= presence background
    # data used for modelling) with training and test data
    # including the meta data
    result_list <- list(pb_data, train_data_with_meta, test_data_with_meta,
        background_test_data)

    # name list contet
    names(result_list) <- c("pb_data", "train_data_with_meta",
        "test_data_with_meta", "background_test_data")

    # save training and test data on disk
    saveRDS(result_list, file = here::here(glue::glue("output/species_data/{species_name}_data.RData")))

    print(glue("Finished subset data of {species_name} after"))

    print(Sys.time() - temp)

    # return results
    return(result_list)

}  # end of function
```

Basic modelling functions

The model building for this study was strongly inspired by the Valavi et al.:

Valavi, R., G. Guillera-Arroita, J. Lahoz-Monfort, and J. Elith. 2021. Predictive performance of presence-only species distribution models: a benchmark study with reproducible code. Ecological Monographs

```r
# function for simultaneous tuning of maxent regularization multiplier and features
maxent_param <- function(data, y = "occ", k = 5, folds = NULL, filepath){
  require(dismo)
  require(caret)
  require(precrec)
  if(is.null(folds)){
    # generate balanced CV folds
    folds <- caret::createFolds(y = as.factor(data$occ), k = k)
  }
  names(data)[which(names(data) == y)] <- "occ"
  covars <- names(data)[which(names(data) != y)]
  # regularization multipliers
  ms <- c(0.5, 1, 2, 3, 4)
  grid <- expand.grid(
    regmult = paste0("betamultiplier=", ms),
    features = list(
      c("noautofeature", "nothreshold"), # LQHP
      c("noautofeature", "nothreshold", "noproduct"), # LQH
      c("noautofeature", "nothreshold", "nohinge", "noproduct"), # LQ
      c("noautofeature", "nothreshold", "nolinear", "noquadratic", "noproduct"), # H
      c("noautofeature", "nothreshold", "noquadratic", "nohinge", "noproduct")), # L
    stringsAsFactors = FALSE
  )

  AUCs <- c()
  for(n in seq_along(grid[,1])){
    if(n%%5 == 0) {print(glue::glue("{n}/{25}"))}
    full_pred <- data.frame()
    for(i in seq_len(length(folds))){
      trainSet <- unlist(folds[-i])
      testSet <- unlist(folds[i])
      if(inherits(try(
        maxmod <- dismo::maxent(x = data[trainSet, covars],
                                p = data$occ[trainSet],
                                removeDuplicates = FALSE,
                                path = filepath,
                                args = as.character(unlist(grid[n, ]))
        )
      ), "try-error")){
        next
      }
      modpred <- predict(maxmod, data[testSet, covars], args = "outputformat=cloglog")
      pred_df <- data.frame(score = modpred, label = data$occ[testSet])
      full_pred <- rbind(full_pred, pred_df)
    }
    AUCs[n] <- precrec::auc(precrec::evalmod(scores = full_pred$score,
                                             labels = full_pred$label))[1,4]
  }
  best_param <- as.character(unlist(grid[which.max(AUCs), ]))
  return(best_param)
}
```

```r
# maxent modelling function
maxent_model <- function (pb_data, env_data_df, covars, species_name)
{
  require(dismo)

  # make sure colum cell us not used for background points
  env_data_df_model <- env_data_df[,names(env_data_df)!= "cell"]


  # number of folds
  nfolds <- ifelse(sum(pb_data$occ) < 10, 2, 5)

  set.seed(32639)
  # tune maxent parameters
  param_optim <- maxent_param(data = pb_data,
                              k = nfolds)


  # fit a maxent model with the tuned parameters
  maxmod <- dismo::maxent(x = pb_data[, covars],
                          p = pb_data$occ,
                          removeDuplicates = FALSE,
                          args = param_optim,
  path = glue::glue(here::here("output/maxent/maxent_{species_name}"))
  )


  # prediction
  maxent_pred  <- dismo::predict(maxmod, env_data_df_model)

  # output
  output <- list(maxmod, maxent_pred)

  return (output)

}

## Gam function ------------------------------------------------------------

gam_model <- function(pb_data, form, env_data_df,covars){


  # normalize env_data with pb_data mean and sd (lasso and maxent do this internally)
  for(v in covars){

    meanv <- mean(pb_data[,v])

    sdv <- sd(pb_data[,v])

    norm_temp_pb <- data.frame((pb_data[[v]] - meanv) / sdv)
    norm_temp_env <- data.frame((env_data_df[[v]] - meanv) / sdv)

    if (!exists("norm_env")){
```

```
      norm_env <- norm_temp_env
      norm_pb <- norm_temp_pb

    } else {
      norm_env <- cbind(norm_env,norm_temp_env)
      norm_pb <- cbind(norm_pb,norm_temp_pb)}

  } # end of loop

  # correct names of normalized variables
  names(norm_env) <- covars
  names(norm_pb) <- covars
  norm_pb$occ <- pb_data$occ



  require(mgcv)
  # calculating the case weights (equal weights)
  # the order of weights should be the same as presences and backgrounds in the training data
  prNum <- as.numeric(table(norm_pb$occ)["1"]) # number of presences
  bgNum <- as.numeric(table(norm_pb$occ)["0"]) # number of backgrounds

  # calculate weights (had to be put as column to pb_data for the gam function to be able to access it)
  norm_pb$weights <- ifelse(norm_pb$occ == 1, 1, prNum/bgNum)

  # gam model
  gam_mod <- mgcv::gam(formula = as.formula(form),
                       data = norm_pb,
                       family = binomial(link = "logit"),
                       weights = weights,
                       method = "REML")

  # predict
  gam_pred <- predict.gam(
    object = gam_mod,
    newdata = norm_env,
    type = "response"
  )

  # output
  output <- list(gam_mod, gam_pred)

  return(output)
}









## Lasso ---------------------------------------------------------------
```

```r
lasso_model <- function (pb_data, form, env_data_df, env_data_rast, covars) {

  # make sure colum cell us not used for background points
  env_data_df_model <- env_data_df[,names(env_data_df)!= "cell"]

  require(myspatial)
  require(glmnet)

  # generating the quadratic terms for all continuous variables
  quad_obj <- myspatial::make_quadratic(pb_data, cols = covars)

  # now we can predict this quadratic object on the training and testing data
  # this makes two columns for each covariate used in the transformation
  training_quad <- predict(quad_obj, newdata = pb_data)
  testing_quad <- predict(quad_obj, newdata = env_data_df_model)



  #convert the data.frames to sparse matrices
  # select all quadratic (and non-quadratic) columns, except the y (occ)
  new_vars <- names(training_quad)[!names(training_quad) %in% c("occ")]
  training_sparse <- Matrix::sparse.model.matrix(~. -1, training_quad[, new_vars])
  testing_sparse <- Matrix::sparse.model.matrix( ~. -1, testing_quad[, new_vars])

  # calculating the case weights
  prNum <- as.numeric(table(training_quad$occ)["1"]) # number of presences
  bgNum <- as.numeric(table(training_quad$occ)["0"]) # number of backgrounds
  wt <- ifelse(pb_data$occ == 1, 1, prNum / bgNum)


  # plot the regularization path and shrinkage in the coefficients
  # plot(lasso, xvar = "lambda", label = TRUE)

  set.seed(32639)

  # cross-validation for parameter estimation
  lasso_cv <- glmnet::cv.glmnet(x = training_sparse,
                        y = training_quad$occ,
                        family = "binomial",
                        alpha = 1, # fitting lasso
                        weights = wt,
                        nfolds = 10, # number of folds for cross-validation
                        standardize = TRUE)


  # predicting with lasso model
  lasso_pred <- stats::predict(lasso_cv,
                               testing_sparse,
                               type = "response",
                               s = "lambda.1se")
```

```
  # output
  output <- list(lasso_cv, lasso_pred)

  return(output)

} # end of function




## RandomForest downsampled ------------------------------------------------


rf_ds_model <- function (pb_data, env_data_df, covars) {
  # make sure colum cell us not used for background points
  env_data_df_model <- env_data_df[,names(env_data_df)!= "cell"]

  # convert the response to factor for producing class relative likelihood
  pb_data_rf <- pb_data

  pb_data_rf$occ <- as.factor(pb_data$occ)

  # set number of presence and bg points
  prNum <- as.numeric(table(pb_data_rf$occ)["1"]) # number of presences
  bgNum <- as.numeric(table(pb_data_rf$occ)["0"]) # number of backgrounds


  # the sample size in each class; the same as presence number
  smpsize <- c("0" = prNum, "1" = prNum)

  # set seed
  set.seed(32639)

  # normalize env_data with pb_data mean and sd (lasso and maxent do this internally)
  for(v in covars){

    meanv <- mean(pb_data_rf[,v])

    sdv <- sd(pb_data_rf[,v])

    norm_temp_pb <- data.frame((pb_data_rf[[v]] - meanv) / sdv)
    norm_temp_env <- data.frame((env_data_df[[v]] - meanv) / sdv)

    if (!exists("norm_env")){

      norm_env <- norm_temp_env
      norm_pb <- norm_temp_pb

    } else {
      norm_env <- cbind(norm_env,norm_temp_env)
      norm_pb <- cbind(norm_pb,norm_temp_pb)}

    } # end of loop
```

```
  # correct names of normalized variables
  names(norm_env) <- covars
  names(norm_pb) <- covars
  norm_pb$occ <- pb_data_rf$occ

  # rf modelling
  rf_downsample <- randomForest::randomForest(formula = occ ~.,
                                data = norm_pb,
                                ntree = 1000,
                                sampsize = smpsize,
                                replace = TRUE)


  # rf prediction
  rf_ds_pred <- stats::predict(rf_downsample,
                               norm_env,
                               type = "prob")

  # output
  output <- list(rf_downsample, rf_ds_pred)

  return(output)
} # end of function
```

Ensemble Modelling

```
# ensemble function
# ----------------------------------------------------------
ensemble <- function(species_name, env_data_df, path_env_data_rast,
    form, clhs_ind, bool_return = F) {

    gc()

    tmp_total <- Sys.time()

    # set up rast data
    env_data_rast <- terra::rast(path_env_data_rast)

    # subset full data set for current species
    # (species_name)

    subset_data <- readRDS(file = here::here(glue::glue("output/species_data/{species_name}_data.RData"))

    # set up variables as given by subset_data
    pb_data <- subset_data[[1]]
    train_data <- subset_data[[2]]
    test_data <- subset_data[[3]]

    # vector containing covariate names
    covars <- colnames(pb_data[, !names(pb_data) %in% c("occ")])
```

```
tmp_model <- Sys.time()

print(glue("Start GAM Modelling of species {species_name}"))

# gam prediction
gam <- gam_model(pb_data = pb_data, env_data_df = env_data_df,
    form = form, covars = covars)

print(glue("Finished GAM Modelling for {species_name}"))
print(Sys.time() - tmp_model)

print("------------------------------------------------------------------------------")

print(glue("Start Lasso Modelling of species {species_name}"))

# time model start
tmp_model <- Sys.time()

# lasso prediction
lasso <- lasso_model(pb_data = pb_data, env_data_df = env_data_df,
    env_data_rast = env_data_rast, form = form, covars = covars)

print(glue("Finished Lasso modelling of species {species_name} after"))
print(Sys.time() - tmp_model)

print("------------------------------------------------------------------------------")

print(glue("Start maxent Modelling of species {species_name}"))

# time model start
tmp_model <- Sys.time()

# maxent prediction
maxent <- maxent_model(pb_data = pb_data, env_data_df = env_data_df,
    covars = covars, species_name = species_name)

print(glue("Finished MaxEnt Modelling of species {species_name}"))
print(Sys.time() - tmp_model)

print("------------------------------------------------------------------------------")

print(glue("Start RandomForest Modelling of species {species_name}"))

# time model start
tmp_model <- Sys.time()

# randomForest downsampled prediction
randomforest <- rf_ds_model(pb_data = pb_data, env_data_df = env_data_df,
    covars = covars)

print(glue("Finished RandomForest Modelling of species {species_name}"))
print(Sys.time() - tmp_model)
```

```
    # safe models
    models <- list(gam[[1]], lasso[[1]], randomforest[[1]], maxent[[1]])
    names(models) <- c("gam", "lasso", "randomforest", "maxent")
    saveRDS(models, file = here::here(glue::glue("built_model/{species_name}_model.RData")))

    # create unweighted average ensemble
    ensemble <- rowMeans(cbind(gam[[2]], lasso[[2]], randomforest[[2]][,
        2], maxent[[2]]))

    # list with predictions
    predictions <- list(gam[[2]], lasso[[2]], randomforest[[2]][,
        2], maxent[[2]], ensemble)

    # name list
    names(predictions) <- c("gam", "lasso", "randomforest", "maxent",
        "ensemble")

    # safe prediction to disk
    saveRDS(predictions, file = here::here(glue::glue("predictions/{species_name}_prediction.RData")))

    print(glue("Finished full modelling for {species_name} after"))
    print(Sys.time() - tmp_total)

    print("--------------------------------------------------------------------------------")
    print("--------------------------------------------------------------------------------")

    gc()

}
```

Function to create prediction raster

```
# prediction rast
# ------------------------------------------------

prediction_rast <- function(prediction, env_data_rast, env_data_df) {

    pred_rast <- terra::rast(nrows = nrow(env_data_rast), ncols = ncol(env_data_rast),
        xmin = xmin(env_data_rast), xmax = xmax(env_data_rast),
        ymin = ymin(env_data_rast), ymax = ymax(env_data_rast),
        crs = "epsg:4326")

    terra::set.values(x = pred_rast, cell = env_data_df$cell,
        values = prediction)

    return(pred_rast)


}
```

Basic Function for Evaluation

```
# Evaluation
# ------------------------------------------------------


# evaluation function for generating a data frame
# containing Mean absolute error, AUC, and Boyceindex
# values
evaluation <- function(prediction, pred_rast, test_data, background_test_data,
    env_data_rast, env_data_df) {

    time_eval <- Sys.time()

    require(Metrics)
    require(mecofun)
    require(sf)
    require(terra)
    require(ecospat)

    # test data spatial object
    test_data_sf <- sf::st_as_sf(test_data, coords = c("x", "y"))

    # extract prediction values for test data
    test_extr = terra::extract(pred_rast, test_data_sf)

    # correct column names and add occurrence column
    colnames(test_extr) <- c("ID", "predcicted")
    test_extr$occ <- 1

    # create spatial feature for random evaluation points
    background_test_data_sf <- sf::st_as_sf(as.data.frame(background_test_data),
        coords = c("x", "y"))

    # extract prediction value for random points
    background_test_data_extr <- terra::extract(pred_rast, background_test_data_sf,
        layer = "lyr.1")

    # correct column names and occurence column
    colnames(background_test_data_extr) <- c("ID", "predcicted")
    background_test_data_extr$occ <- 0

    # bind evaluation data
    eval_data <- rbind(test_extr, background_test_data_extr)

    # merics
    auc <- Metrics::auc(actual = eval_data$occ, predicted = eval_data$predcicted)
    mae <- Metrics::mae(actual = eval_data$occ, predicted = eval_data$predcicted)
    rmse <- Metrics::rmse(actual = eval_data$occ, predicted = eval_data$predcicted)
    boyce_index = ecospat::ecospat.boyce(fit = as.vector(prediction),
        obs = eval_data$predcicted[eval_data$occ == 1], PEplot = FALSE)

    thresh_eval <- mecofun::evalSDM(observation = eval_data$occ,
```

```
        predictions = eval_data$predcicted, thresh.method = "MaxSens+Spec")

    print("Finished Evaluation")
    print(Sys.time() - time_eval)
    print("-------------------------------------")

    return(data.frame(auc = auc, mae = mae, rmse = rmse, boyce_index = boyce_index$cor,
        thresh = thresh_eval$thresh, TSS = thresh_eval$TSS))


}
```

Function for Evaluation for all models over an array of species

This code will - load the predictions and transform than into a raster - load test data and training data - evaluate the prediction for both training and test data - safe the output

You can specify if PA1 or PA10 was used.

```
# evaluation and creating raster for all predictions of one
# species

ensemble_eval_full <- function(species_name, env_data_df, path_env_data_rast,
    is_pa1 = T) {

    tmp_eval <- Sys.time()

    print(glue("starting evauation of {species_name}"))

    # read predictions for species x
    predictions <- readRDS(file = glue::glue("output/predictions/{species_name}_prediction.RData"))

    # # set up env_data_rast
    env_data_rast <- terra::rast(path_env_data_rast)
    # # subset full data set for current species
    # (species_name)
    subset_data <- readRDS(file = glue::glue("output/species_data/{species_name}_data.RData"))

    # set up variables as given by subset_data
    training_data <- subset_data[[2]]

    test_data <- subset_data[[3]]
    background_test_data <- subset_data[[4]]

    # create number of PAs either 10 times presences or 1
    # time the presences maximum of 2000 Pas was set in the
    # subset_data function
    if (is_pa1) {
        background_test_data <- background_test_data[1:(1 * nrow(test_data)),
            ]
    } else background_test_data <- background_test_data[1:(10 *
        nrow(test_data)), ]
```

```r
    # run loop over all predictions inside one prediction
    # object
    for (j in 1:length(predictions)) {

        # set up name of model used for prediction and
        # vector of prediction
        pred_name <- names(predictions)[j]

        print(pred_name)

        # get rast
        pred_rast <- prediction_rast(prediction = as.vector(predictions[[j]]),
            env_data_rast = env_data_rast, env_data_df = env_data_df)

        # prediction rast stack
        if (j == 1) {
            pred_rast_stack <- pred_rast
        } else {
            pred_rast_stack <- rast(list(pred_rast_stack, pred_rast))
        }

        # safe evaluation metrics in
        eval <- evaluation(prediction = as.vector(predictions[[j]]),
            pred_rast = pred_rast, test_data = test_data, background_test_data = background_test_data,
            env_data_rast = env_data_rast, env_data_df = env_data_df)
        # create data frame with column names
        eval_df_temp <- data.frame(pred = pred_name, AUC = eval$auc,
            MAE = c(eval$mae), rmse = eval$rmse, boyceIndex = eval$boyce_index,
            threshhold = eval$thresh, TSS = eval$TSS)

        # fill data frame
        if (j == 1) {
            eval_df <- eval_df_temp
        } else {
            eval_df[j, ] <- eval_df_temp
        }

    }  # end of evaluation for loop

    # set up vector for prediction names
    names_species_prediction <- paste0(names(predictions), glue("_{species_name}"))

    # set names of rast stack
    terra::set.names(pred_rast_stack, names_species_prediction)

    # wrap for compatability with parallisation
    pred_rast_stack <- wrap(pred_rast_stack)

    # check which pa was used and safe data accordingly
    if (is_pa1) {
        saveRDS(eval_df, file = here::here(glue::glue("output/evaluation_same/{species_name}_eval_same.
    } else {
        saveRDS(eval_df, file = here::here(glue::glue("output/evaluation_10/{species_name}_eval_10.RData
```

```
}

print(glue("Finished  evaluation for {species_name} after"))
print(Sys.time() - tmp_eval)

background_test_data <- subset_data[[4]]

# same number of PAs as training data
background_test_data <- background_test_data[1:(nrow(training_data)),
    ]

# evaluation training
# ----------------------------------------------------

# run loop over all predictions inside one prediction
# object
for (j in 1:length(predictions)) {

    # set up name of model used for prediction and
    # vector of prediction
    pred_name_training <- names(predictions)[j]

    print(pred_name_training)

    # get rast
    pred_rast_training <- prediction_rast(prediction = as.vector(predictions[[j]]),
        env_data_rast = env_data_rast, env_data_df = env_data_df)

    # safe evaluation metrics in
    eval_training <- evaluation(prediction = as.vector(predictions[[j]]),
        pred_rast = pred_rast_training, test_data = training_data,
        background_test_data = background_test_data, env_data_rast = env_data_rast,
        env_data_df = env_data_df)

    # create data frame with column names
    eval_df_temp_training <- data.frame(pred = pred_name_training,
        AUC = eval_training$auc, MAE = eval_training$mae,
        rmse = eval_training$rmse, boyceIndex = eval_training$boyce_index)

    # fill data frame
    if (j == 1) {
        eval_df_training <- eval_df_temp_training
    } else {
        eval_df_training[j, ] <- eval_df_temp_training
    }

}  # end of evaluation for loop

# check which pa was used and safe data accordingly
if (is_pa1) {
    saveRDS(eval_df, file = here::here(glue::glue("output/evaluation_training_same/{species_name}_ev
} else {
    saveRDS(eval_df, file = here::here(glue::glue("output/evaluation_training_10/{species_name}_eval
```

```
    }

    print(glue("Finished  evaluation for {species_name} after"))
    print(Sys.time() - tmp_eval)




}
```

Processing: Data - Modelling - Evaluation

```
# model form for gam, no interactions, thin plate
# regression spline used on all predictors
form <- occ ~ s(Bio_11_Mean_Temperature_of_Coldest_Quarter) +
    s(Bio_4_Temperature_Seasonality) + +s(Bio_2_Mean_Diurnal_Range) +
    s(Bio_17_Precipitation_of_Driest_Quarter) + +s(Bio_18_Precipitation_of_Warmest_Quarter) +
    s(Bio_19_Precipitation_of_Coldest_Quarter) + +s(Bio_16_Precipitation_of_Wettest_Quarter) +
    s(Bio_15_Precipitation_Seasonality)




# select species for modelling (>= 10 occurences)
species_10 <- occ_data_clean %>%
    dplyr::group_by(species) %>%
    dplyr::filter(dplyr::n() >= 10) %>%
    dplyr::select(species) %>%
    unique() %>%
    dplyr::pull()







# initialize parallel computing
# ------------------------------------------------


# initalize parallel computing with snowfall
sfInit(parallel = TRUE, cpus = 6, slaveOutfile = "logfile.txt")

# java memory setup (otherwise Maxent might produce a
# meomry error
#- has to be run as the first command of the session)
snowfall::sfClusterEval(options(java.parameters = "-Xmx8g"))
```

```r
# export variables needed for modelling
sfExport("occ_data_clean")
sfExport("species_10")
sfExport("env_data_df")
sfExport("form")
sfExport("clhs_index")
sfExport("%not_in%")



# Load Modelling functions
sfSource(here::here("process/scripts/modelling/modelling_functions.R"))


# load packages for parallel computing
lapply(packages, sfLibrary, character.only = TRUE)



# data
# ------------------------------------------------------------------


# subset data for all species in parallel running (writes
# data to disk)
sfLapply(1:length(species_10), function(x) {

    subset_data <- subset_species(data = occ_data_clean, species_name = species_10[x],
        path_env_data_rast = here("data/processed_data/final/env_data_rast.tif"),
        env_data_df = env_data_df, clhs_ind = clhs_index)
})



# model
# ------------------------------------------------------------------

snowfall::sfClusterApply()

# ensemble model for all species in parallel running
# (writes data to disk)
sfLapply(1:length(species_10), function(x) {

    gc()

    ensemble(env_data_df = env_data_df, species_name = species_10[x],
        path_env_data_rast = here("data/processed_data/final/env_data_rast.tif"),
        form = form, clhs_ind = clhs_index)

    gc()


}  # end of code input(function)
```

```
)  # end of apply




# evaluation
# ------------------------------------------------------------


# ensemble model for all species in parallel running
# (writes data to disk)
sfLapply(1:length(species_10), function(x) {

    gc()

    ensemble_eval(species_name = species_10[x], env_data_df = env_data_df,
        path_env_data_rast = here("data/processed_data/final/env_data_rast.tif"))

    gc()


}  # end of code input(function)
)  # end of apply
```

Processing evaluation data for PA1

In the following code all evaluation data will be analysed for PA1.

- Mean performances will be calculated for all threshholds
- Ranks will be calculated for species with more than 20 occurences

```
# read and combine all prediction raster and evaluation data --------------------------------


# setup data for for loop
eval_list <- list()
pred_rast_list <- list()
if(exists("sum_pred_rast")) {rm(sum_pred_rast)}
target_crs <- st_crs("EPSG:4326")


# loop over all species used
for (i in 1:length(species_10)) {


    # print out
    temp <-Sys.time()
    print("Starting")
```

```
    print(glue("{i}/{length(species_10)}:{species_10[i]}"))

    # read in evaluation and raster data (of all 5 predictions for one species)
    eval <- readRDS(here::here(glue::glue("output/evaluation_same/{species_10[i]}_eval_same.RData")))
    rank <- rank(eval$boyceIndex)

    # append evaluation data frame to list
    eval_list <- append(eval_list, list(cbind(eval,rank)))

    # read prediction spatRaster
    rast <- readRDS(here::here(glue::glue("output/raster/{species_10[i]}_rast.RData")))



    # sum of spatRasters
    if(i == 1) {sum_pred_rast <- rast
    } else {sum_pred_rast <- sum_pred_rast + rast}




    print(Sys.time() - temp)

} # end of for loop




# for training ------------------------------------------------------------



# read and combine all prediction raster and evaluation data --------------------------------


# setup data for for loop

eval_list_training <- list()

# loop over all species used
for (i in 1:length(species_10)) {

  if (i!=60){
    # print out
    temp <-Sys.time()
    print("Starting")
    print(glue("{i}/{length(species_10)}:{species_10[i]}"))

    # read in evaluation and raster data (of all 5 predictions for one species)
    eval <- readRDS(here::here(
```

```
      glue::glue("output/evaluation_training_same/{species_10[i]}_eval_training_same.RData")))
    rank <- rank(eval$boyceIndex)

    # append evaluation data frame to list
    eval_list_training <- append(eval_list_training, list(cbind(eval,rank)))


  }
} # end of for loop



names(eval_list_training) <- species_10









# create columns indicating number of presences for evaluating threshholds ----------------------------
spe_10_with_occs <- occ_data_clean %>%
  group_by(species) %>%
  dplyr::mutate(occurences = n()) %>%
  group_by(species) %>%
  dplyr::filter(n() >=10) %>%
  dplyr::mutate(bigger_30  = n() >=30) %>%
  dplyr::mutate(bigger_100  = n() >=100)%>%
  dplyr::mutate(bigger_20  = n() >=20) %>%
  dplyr::mutate(smaller_30  = n() <30)

final_check_species <- unique(spe_10_with_occs[,c(1,9,10,11,12)])









# training data evaluation ----------------------------------------------------


# -----------------------------------------------------------------------

# mean of list of dataframes 10 ----------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_training <- lapply(eval_list_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
```

```r
mean_df <- data.frame(matrix(ncol = 4, nrow = 5))
names(mean_df) <- names(df_list_reduced_training[[1]])
median_df <- mean_df
sd_df <- mean_df
se_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_training, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the results to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the resulting data frames in a list
result_training <- list("Mean" = mean_df, "Median" = median_df, "se" = se_df)

# print out the results
print(result_training)







# ------------------------------------------------------------------------

final_check_species

table(final_check_species$smaller_30)
eval_list_10 <- eval_list[names(eval_list) %in% final_check_species$species[final_check_species$smaller
length(eval_list_10)

# mean of list of dataframes 10 ----------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_10 <- lapply(eval_list, function(df) df[, 2:7])
# df_list_reduced <- lapply(eval_list_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_10[[1]])
median_df <- mean_df
```

```r
sd_df <- mean_df
se_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_10, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the results to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the resulting data frames in a list
result_10 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, "se" = se_df)

# print out the results
print(result_10)









# 20 ------------------------------------------------------------------------


# ---------------------------------------------------------------------

final_check_species

table(final_check_species$bigger_20)
eval_list_20 <- eval_list[names(eval_list) %in% final_check_species$species[final_check_species$bigger_
length(eval_list_20)

# mean of list of dataframes 20 ---------------------------------------------------------

# keep only the numeric columns
```

```r
df_list_reduced_20 <- lapply(eval_list_20, function(df) df[, 2:7])
# df_list_reduced <- lapply(eval_list_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_20[[1]])
median_df <- mean_df
sd_df <- mean_df
se_df <-  mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_20, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the results to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the resulting data frames in a list
result_20 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, "SE" = se_df, se = se_df)

# print out the results
print(result_20)




# 30 ----------------------------------------------------------------------------



# ----------------------------------------------------------------------------


final_check_species

table(final_check_species$bigger_30)
eval_list_30 <- eval_list[
  names(eval_list) %in% final_check_species$species[final_check_species$bigger_30]
  ]
length(eval_list_30)
```

```r
# mean of list of dataframes 30 ------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_30 <- lapply(eval_list_30, function(df) df[, 2:7])
# df_list_reduced <- lapply(eval_list_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_30[[1]])
median_df <- mean_df
sd_df <- mean_df
sd_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_30, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them
  # in mean_df, median_df, and sd_df
  # Use round() function to round the results to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2,
                            FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))), 3)

  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the resulting data frames in a list
result_30 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, se = se_df)

# print out the results
print(result_30)




# 100 ---------------------------------------------------------------------------



# ------------------------------------------------------------------------


# ------------------------------------------------------------------------
```

```r
final_check_species

table(final_check_species$bigger_100)

eval_list_100 <- eval_list[
  names(eval_list) %in% final_check_species$species[final_check_species$bigger_100]
  ]

length(eval_list_100)


# mean of list of dataframes 100 ------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_100 <- lapply(eval_list_100, function(df) df[, 2:7])
# df_list_reduced <- lapply(eval_list_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_100[[1]])
median_df <- mean_df
sd_df <- mean_df
sd_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the results to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),

  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the resulting data frames in a list
results_100 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, se = se_df)

# print out the results
print(results_100)
```

```r
# Ranks for threshhold 20 ------------------------------------------------------------------

df_list_reduced <- df_list_reduced_20

# Loop over each row
for (i in 1:length(df_list_reduced)) {


  df_list_reduced[[i]]$boyce_rank <- rank(df_list_reduced[[i]]$boyceIndex,ties.method= "max" ) %>%
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)

  df_list_reduced[[i]]$auc_rank <- rank(df_list_reduced[[i]]$AUC,ties.method= "max" ) %>%
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)



  df_list_reduced[[i]]$TSS_rank <- rank(df_list_reduced[[i]]$TSS,ties.method= "max" ) %>%
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)




  df_list_reduced[[i]]$MAE_rank <- rank(df_list_reduced[[i]]$MAE,ties.method= "max" )
  df_list_reduced[[i]]$rmse_rank <- rank(df_list_reduced[[i]]$rmse,ties.method= "max" )
```

```
}

rank_list_boyce <- list()
rank_list_auc <- list()
rank_list_MAE <- list()
rank_list_rmse <- list()
rank_list_TSS <- list()

# boyce ------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_boyce[i] <- do.call(rbind, rows) %>%
    group_by(boyce_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_boyce)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_boyce

for (i in 1:length(rank_list_boyce)) {
  print(weighted.mean(rank_list_boyce[[i]][1],rank_list_boyce[[i]][3]))
}

# auc --------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced, function(df) df[i, ])
  rank_list_auc[i] <- do.call(rbind, rows) %>%
    group_by(auc_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()


}
names(rank_list_auc)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_auc
```

```r
# New row to be added
new_row <- data.frame(
  auc_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)

rank_list_auc$Ensemble <-rbind(rank_list_auc$Ensemble, new_row)

rank_list_auc$Ensemble



for (i in 1:length(rank_list_auc)) {
  print(weighted.mean(rank_list_auc[[i]][1],rank_list_auc[[i]][3]))
}

# MAE ----------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_MAE[i] <- do.call(rbind, rows) %>%
    group_by(MAE_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_MAE)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_MAE




# New row to be added
new_row_1 <- data.frame(
  MAE_rank      = 1,
  n = 0,  # example value
  percentage = 0  # example value
)
# New row to be added
new_row_5 <- data.frame(
  MAE_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)
```

```r
rank_list_MAE$Ensemble <-rbind(new_row_1,rank_list_MAE$Ensemble,new_row_5)

rank_list_MAE$Ensemble


for (i in 1:length(rank_list_MAE)) {
  print(weighted.mean(rank_list_MAE[[i]][1],rank_list_MAE[[i]][3]))
}



# TSS ----------------------------------------------------------------------

rank_list_TSS <- list()
# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced, function(df) df[i, ])


  # Combine these rows into a new data frame
  rank_list_TSS[i] <- do.call(rbind, rows) %>%
    group_by(TSS_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_TSS)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_TSS


for (i in 1:length(rank_list_TSS)) {
  print(weighted.mean(rank_list_TSS[[i]][1],rank_list_TSS[[i]][3]))
}




# RMSE ----------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_rmse[i] <- do.call(rbind, rows) %>%
    group_by(rmse_rank) %>%
    count() %>%
    ungroup() %>%
```

```
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_rmse)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_rmse

# New row to be added
new_row_5 <- data.frame(
  rmse_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)

rank_list_rmse$Ensemble <-rbind(rank_list_rmse$Ensemble,new_row_5)


for (i in 1:length(rank_list_rmse)) {
  print(weighted.mean(rank_list_rmse[[i]][1],rank_list_rmse[[i]][3]))
}
```

Processing evaluation data for PA10

Here the same as before will be done for PA1

```
# read and combine all prediction raster and evaluation data --------------------------------


# setup data for for loop
eval_list_100 <- list()
pred_rast_list <- list()
if(exists("sum_pred_rast")) {rm(sum_pred_rast)}
target_crs <- st_crs("EPSG:4326")


# loop over all species used
for (i in 1:length(species_10)) {

  # print out
  temp <-Sys.time()
  print("Starting")
  print(glue("{i}/{length(species_10)}:{species_10[i]}"))

  # read in evaluation and raster data (of all 5 predictions for one species)
  eval <- readRDS(here::here(
    glue::glue("output/evaluation_10/{species_10[i]}_eval_10.RData")))
  rank <- rank(eval$boyceIndex)

  # append evaluation data frame to list
  eval_list <- append(eval_list, list(cbind(eval,rank)))

  # read prediction spatRaster
```

```
  rast <- readRDS(here::here(glue::glue("output/raster/{species_10[i]}_rast.RData")))



  # sum of spatRasters
  if(i == 1) {sum_pred_rast <- rast
  } else {sum_pred_rast <- sum_pred_rast + rast}




  print(Sys.time() - temp)

} # end of for loop



# eval_list_100 <- readRDS("D:/plots_10_09/eval_list_10_all_313.RData")
names(eval_list_100) <- species_10[-60]
# for training ------------------------------------------------------------



# read and combine all prediction raster and evaluation data --------------------------------


# setup data for for loop

eval_list_training <- list()

# loop over all species used
for (i in 1:length(species_10)) {

  if (i!=60){
    # print out
    temp <-Sys.time()
    print("Starting")
    print(glue("{i}/{length(species_10)}:{species_10[i]}"))

    # read in evaluation and raster data (of all 5 predictions for one species)
    eval <- readRDS(here::here(
      glue::glue("output/evaluation_training_10/{species_10[i]}_eval_training_10.RData")))
    rank <- rank(eval$boyceIndex)

    # append evaluation data frame to list
    eval_list_training <- append(eval_list_training, list(cbind(eval,rank)))


  }
} # end of for loop
```

```r
spe_10_with_occs <- occ_data_clean %>%
  group_by(species) %>%
  dplyr::mutate(occurences = n()) %>%
  group_by(species) %>%
  dplyr::filter(n() >=10) %>%

  dplyr::mutate(bigger_30  = n() >=30) %>%
  dplyr::mutate(bigger_100  = n() >=100)%>%
  dplyr::mutate(bigger_20  = n() >=20) %>%
  dplyr::mutate(smaller_30  = n() <30)

final_check_species <- unique(spe_10_with_occs[,c(1,9,10,11,12)])


# calculate mean of predictions
# pred_rast_mean <- sum_pred_rast/length(species_10)
# saveRDS(pred_rast_mean, "output/pred_rast_mean")



# old ---------------------------------------------------------------------
```

```r
# training evaluation -----------------------------------------------------


# -------------------------------------------------------------------------

table(final_check_species$bigger_20)
eval_list_100_training <- eval_list_100_training[names(eval_list_100_training) %in% final_check_species$
length(eval_list_100_training)

# mean of list of dataframes 10 -------------------------------------------------------------

# keep only the numeric columns
```

```r
df_list_reduced_100_training <- lapply(eval_list_100_training, function(df) df[, 2:5])
# df_list_reduced_100 <- lapply(eval_list_100_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 4, nrow = 5))
names(mean_df) <- names(df_list_reduced_100_training[[1]])
median_df <- mean_df
sd_df <- mean_df
se_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100_training, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the result_100s to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2,
                            FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))), 3)
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the result_100ing data frames in a list
result_100_training <- list("Mean" = mean_df, "Median" = median_df, "se" = se_df)

# print out the result_100s
print(result_100_training)


#
#



# ------------------------------------------------------------------------

final_check_species

table(final_check_species$smaller_30)

eval_list_100_10 <- eval_list_100[
  names(eval_list_100) %in% final_check_species$species[final_check_species$smaller_30]
  ]

length(eval_list_100_10)
```

```r
# mean of list of dataframes 10 ------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_100_10 <- lapply(eval_list_100, function(df) df[, 2:7])
# df_list_reduced_100 <- lapply(eval_list_100_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_100_10[[1]])
median_df <- mean_df
sd_df <- mean_df
se_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100_10, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the result_100s to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the result_100ing data frames in a list
result_100_10 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, "se" = se_df)

# print out the result_100s
print(result_100_10)









# 20 ---------------------------------------------------------------------------


# ------------------------------------------------------------------------------
```

```r
final_check_species

table(final_check_species$bigger_20)

eval_list_100_20 <- eval_list_100[
  names(eval_list_100) %in% final_check_species$species[final_check_species$bigger_20]
  ]

length(eval_list_100_20)

# mean of list of dataframes 20 ------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_100_20 <- lapply(eval_list_100_20, function(df) df[, 2:7])
# df_list_reduced_100 <- lapply(eval_list_100_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_100_20[[1]])
median_df <- mean_df
sd_df <- mean_df
se_df <-  mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100_20, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the result_100s to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the result_100ing data frames in a list
result_100_20 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, "SE" = se_df)

# print out the result_100s
print(result_100_20)




# 30 ---------------------------------------------------------------------------
```

```r
# ------------------------------------------------------------------------


final_check_species


table(final_check_species$bigger_30)

eval_list_100_30 <- eval_list_100[
  names(eval_list_100) %in% final_check_species$species[final_check_species$bigger_30]
  ]

length(eval_list_100_30)

# mean of list of dataframes 30 ------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_100_30 <- lapply(eval_list_100_30, function(df) df[, 2:7])
# df_list_reduced_100 <- lapply(eval_list_100_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_100_30[[1]])
median_df <- mean_df
sd_df <- mean_df
sd_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100_30, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for each column in row_df and store them in mean_df, median_df, an
  # Use round() function to round the result_100s to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),

  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the result_100ing data frames in a list
result_100_30 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, se = se_df)

# print out the result_100s
```

```r
print(result_100_30)



# 100 ------------------------------------------------------------------------



# ----------------------------------------------------------------------------


# ----------------------------------------------------------------------------


final_check_species

table(final_check_species$bigger_100)

eval_list_100_100 <- eval_list_100[
  names(eval_list_100) %in% final_check_species$species[final_check_species$bigger_100]
  ]
length(eval_list_100_100)


# mean of list of dataframes 100 ---------------------------------------------------------------

# keep only the numeric columns
df_list_reduced_100_100 <- lapply(eval_list_100_100, function(df) df[, 2:7])
# df_list_reduced_100 <- lapply(eval_list_100_training, function(df) df[, 2:5])

# Create three empty data frames to store the mean, median, and sd values
mean_df <- data.frame(matrix(ncol = 6, nrow = 5))
names(mean_df) <- names(df_list_reduced_100_100[[1]])
median_df <- mean_df
sd_df <- mean_df
sd_df <- mean_df

# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  row_df <- do.call(rbind, rows)

  # Compute the mean, median, and sd for
  # each column in row_df and store them in mean_df, median_df, and sd_df
  # Use round() function to round the result_100s to 3 decimal places
  mean_df[i, ] <- round(colMeans(row_df, na.rm = TRUE), 3)
  median_df[i, ] <- round(apply(row_df, 2, median, na.rm = TRUE), 3)
  sd_df[i, ] <- round(apply(row_df, 2, sd, na.rm = TRUE), 3)
  se_df[i, ] <- round(apply(row_df, 2, FUN = function(x) sd(x, na.rm = TRUE)/sqrt(length(na.omit(x)))),
```

```
  row.names(mean_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(median_df) <-c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
  row.names(sd_df) <- c("GAM","Lasso","randomforest" ,"Maxent", "ensemble")
}

# Store the result_100ing data frames in a list
result_100s_100 <- list("Mean" = mean_df, "Median" = median_df, "SD" = sd_df, se = se_df)

# print out the result_100s
print(result_100s_100)









# Ranks ------------------------------------------------------------------

df_list_reduced_100 <- df_list_reduced_100_20

# Loop over each row
for (i in 1:length(df_list_reduced_100)) {


  df_list_reduced_100[[i]]$boyce_rank <- rank(df_list_reduced_100[[i]]$boyceIndex) %>%
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)

  df_list_reduced_100[[i]]$auc_rank <- rank(df_list_reduced_100[[i]]$AUC) %>%
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)

  df_list_reduced_100[[i]]$TSS_rank <- rank(df_list_reduced_100[[i]]$TSS) %>%
```

```
    dplyr::case_match(5 ~ 1,
                      4 ~ 2,
                      3 ~ 3,
                      2 ~ 4,
                      1 ~ 5)

  df_list_reduced_100[[i]]$MAE_rank <- rank(df_list_reduced_100[[i]]$MAE)
  df_list_reduced_100[[i]]$rmse_rank <- rank(df_list_reduced_100[[i]]$rmse)



}

rank_list_100_boyce <- list()
rank_list_100_auc <- list()
rank_list_100_MAE <- list()
rank_list_100_rmse <- list()
rank_list_100_TSS <- list()

# boyce ------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_100_boyce[i] <- do.call(rbind, rows) %>%
    group_by(boyce_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_100_boyce)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_100_boyce

for (i in 1:length(rank_list_100_boyce)) {
  print(weighted.mean(rank_list_100_boyce[[i]][1],rank_list_100_boyce[[i]][3]))
}

# auc --------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_100_auc[i] <- do.call(rbind, rows) %>%
```

```
    group_by(auc_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_100_auc)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_100_auc


# New row to be added
new_row <- data.frame(
  auc_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)

rank_list_100_auc$Ensemble <-rbind(rank_list_100_auc$Ensemble, new_row)

rank_list_100_auc$Ensemble



for (i in 1:length(rank_list_100_auc)) {
  print(weighted.mean(rank_list_100_auc[[i]][1],rank_list_100_auc[[i]][3]))
}

# MAE ----------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_100_MAE[i] <- do.call(rbind, rows) %>%
    group_by(MAE_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_100_MAE)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_100_MAE




# New row to be added
new_row_1 <- data.frame(
```

```
  MAE_rank      = 1,
  n = 0,  # example value
  percentage = 0  # example value
)
# New row to be added
new_row_5 <- data.frame(
  MAE_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)

rank_list_100_MAE$Ensemble <-rbind(new_row_1,rank_list_100_MAE$Ensemble,new_row_5)
rank_list_100_MAE$GAM <-rbind(rank_list_100_MAE$GAM,new_row_5)
rank_list_100_MAE$Lasso <-rbind(new_row_1,rank_list_100_MAE$Lasso)
rank_list_100_MAE$Random_Forest_downsampled <-rbind(new_row_1,
                                                    rank_list_100_MAE$Random_Forest_downsampled)

rank_list_100_MAE


for (i in 1:length(rank_list_100_MAE)) {
  print(weighted.mean(rank_list_100_MAE[[i]][1],rank_list_100_MAE[[i]][3]))
}



# TSS ----------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_100_TSS[i] <- do.call(rbind, rows) %>%
    group_by(TSS_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_100_TSS)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_100_TSS


for (i in 1:length(rank_list_100_TSS)) {
  print(weighted.mean(rank_list_100_TSS[[i]][1],rank_list_100_TSS[[i]][3]))
}
```

```r
# RMSE ------------------------------------------------------------------


# Loop over each row
for (i in 1:5) {
  # Extract the ith row from each data frame in the list
  rows <- lapply(df_list_reduced_100, function(df) df[i, ])

  # Combine these rows into a new data frame
  rank_list_100_rmse[i] <- do.call(rbind, rows) %>%
    group_by(rmse_rank) %>%
    count() %>%
    ungroup() %>%
    mutate(percentage = n/sum(n)) %>%
    list()

}
names(rank_list_100_rmse)<- c("GAM","Lasso","Random_Forest_downsampled","Maxent", "Ensemble")
rank_list_100_rmse

# New row to be added
new_row_5 <- data.frame(
  rmse_rank      = 5,
  n = 0,  # example value
  percentage = 0  # example value
)

# New row to be added
new_row_2 <- data.frame(
  rmse_rank      = 2,
  n = 0,  # example value
  percentage = 0  # example value
)

rank_list_100_rmse$Ensemble <-rbind(rank_list_100_rmse$Ensemble,new_row_5)


for (i in 1:length(rank_list_100_rmse)) {
  print(weighted.mean(rank_list_100_rmse[[i]][1],rank_list_100_rmse[[i]][3]))
}
```

Plotting and Analyzing evaluation

Heattable for ranks

```r
# heattable PA1 ---------------------------------------------------------------

data_ranks_boyce <- bind_rows(rank_list_boyce, .id = "model")
```

```r
data_ranks_auc <- bind_rows(rank_list_auc, .id = "model")
data_ranks_MAE <- bind_rows(rank_list_MAE, .id = "model")
data_ranks_rmse <- bind_rows(rank_list_rmse, .id = "model")
data_ranks_TSS <- bind_rows(rank_list_TSS, .id = "model")


data_ranks_final <- data.frame(metric= c(rep("boyce",25),rep("auc",25),rep("MAE",25), rep("TSS",25),rep
model= c(data_ranks_boyce$model,data_ranks_auc$model,
         data_ranks_MAE$model, data_ranks_TSS$model, data_ranks_rmse$model),
performance_rank = rep(c(1,2,3,4,5),25),
value = c(
  round(data_ranks_boyce$percentage*100,1),
  round(data_ranks_auc$percentage*100,1),
  round(data_ranks_MAE$percentage*100,1),
  round(data_ranks_TSS$percentage*100,1),
  round(data_ranks_rmse$percentage*100,1)
)
)




data_ranks_final$metric <- factor(data_ranks_final$metric,
                                  levels = c("boyce","MAE","rmse", "auc","TSS"),
                                  labels = c("CBI","MAE","RMSE", "AUC","TSS PA1"))



# Reorder the factor levels for the desired order
data_ranks_final$model <- factor(data_ranks_final$model, levels = c("GAM", "Lasso", "Maxent", "Random_F


data_ranks_final_reduced_same <- data_ranks_final[data_ranks_final$metric%in%c("CBI","TSS PA1"),]

data_ranks_both <- rbind(data_ranks_final_reduced_same,data_ranks_final_reduced_100)

# Example of changing y-axis labels
new_labels <- c("Rank 1", "Rank 2", "Rank 3","Rank 4", "Rank 5")  # Replace with your desired labels
new_breaks <- c(1, 2, 3,4,5)  # Replace with the positions of your labels

heattable_same <- ggplot(data_ranks_both, aes(x = model, y = performance_rank, fill = value)) +
  geom_tile(color = "white", size = 0.5) +
  facet_grid(rows = vars(metric), scales = "free", space = "free_y") +  # Adjust space for y-axis
  # scale_fill_viridis_c(option = "cividis", direction = -1) +  # Use the YlGnBu palette for continuous
  scale_fill_distiller(palette = "YlGnBu", direction = 1) +  # Use the YlGnBu palette for continuous da
  scale_x_discrete(labels = c("Maxent" = "Maxent", "Random_Forest_downsampled" = "Random Forest", "Lasso
                              "Gam" = "Gam", "Ensemble" = "Ensemble")) +
  scale_y_continuous(breaks = new_breaks, labels = new_labels) +  # Modify y-axis labels
  # scale_x_discrete(labels = c("Gam","Lasso","RF","Maxent","Ensemble")) +  # Modify y-axis labels
  labs(title = NULL, x = NULL, y = NULL) +  # Removed title and axis labels
  theme_minimal() +
  geom_text(aes(label = round(value, 2),color = ifelse(value > 30, "white", "black")), size = 1.8) +
  scale_color_manual(values = c("black", "white"))+
```

```
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 6, face = "bold"),
    axis.text.y = element_text(size = 5, hjust = 3),  # Adjust the size here
    strip.background = element_blank(),
    strip.text = element_text(size = 6,face = "bold"),
    strip.placement = "outside",  # Place the facet label to the left
    legend.position = "none",# Remove the legend
    panel.border = element_rect(color = "black", fill = NA, size = 1),  # Add border around each facet)
    panel.spacing = unit(0.2, "cm"),  # Adjust the space between facets. Change "2" to your desired spa
    axis.ticks.y = element_line(color = "black", size = 0.5),
    axis.ticks.length = unit(0.1, "cm"),  # Adjust the tick length
    axis.ticks.x = element_line(color = "black", size = 0.5))
heattable_same


ggsave(here::here("output/plots/heattable_PA1.jpg"), plot = heattable_same, width = 6, height = 4)




# heattable PA10 ------------------------------------------------------------

data_ranks_boyce_100 <- bind_rows(rank_list_100_boyce, .id = "model")
data_ranks_auc_100 <- bind_rows(rank_list_100_auc, .id = "model")
data_ranks_MAE_100 <- bind_rows(rank_list_100_MAE, .id = "model")
data_ranks_rmse_100 <- bind_rows(rank_list_100_rmse, .id = "model")
data_ranks_TSS_100 <- bind_rows(rank_list_100_TSS, .id = "model")


data_ranks_final_100 <- data.frame(metric= c(rep("boyce",25),rep("auc",25),rep("MAE",25), rep("TSS",25)
                                model= c(data_ranks_boyce_100$model,data_ranks_auc_100$model,
                                         data_ranks_MAE_100$model, data_ranks_TSS_100$model, data_ranks_
                                performance_rank = rep(c(1,2,3,4,5),25),
                                value = c(
                                  round(data_ranks_boyce_100$percentage*100,1),
                                  round(data_ranks_auc_100$percentage*100,1),
                                  round(data_ranks_MAE_100$percentage*100,1),
                                  round(data_ranks_TSS_100$percentage*100,1),
                                  round(data_ranks_rmse_100$percentage*100,1)
                                )
)

# Sample data
set.seed(123)


data_ranks_final_100$metric <- factor(data_ranks_final_100$metric,
                                   levels = c("boyce","MAE","rmse", "auc","TSS"),
                                   labels = c("CBI_10","MAE_10","RMSE_10","AUC_10","TSS PA10"))


# Reorder the factor levels for the desired order
```

```r
data_ranks_final_100$model <- factor(data_ranks_final_100$model, levels = c("GAM", "Lasso", "Maxent", "


data_ranks_final_reduced_100 <- data_ranks_final_100[data_ranks_final_100$metric%in%c("TSS PA10"),]
# Example of changing y-axis labels
new_labels <- c("Rank 1", "Rank 2", "Rank 3","Rank 4", "Rank 5")  # Replace with your desired labels
new_breaks <- c(1, 2, 3,4,5)  # Replace with the positions of your labels

heattable_100 <- ggplot(data_ranks_final_reduced_100, aes(x = model, y = performance_rank, fill = value)
  geom_tile(color = "white", size = 0.5) +
  facet_grid(rows = vars(metric), scales = "free", space = "free_y") +  # Adjust space for y-axis
  # scale_fill_viridis_c(option = "cividis", direction = -1) +  # Use the YlGnBu palette for continuous
  scale_fill_distiller(palette = "YlGnBu", direction = 1) +  # Use the YlGnBu palette for continuous da
  scale_x_discrete(labels = c("Maxent" = "Maxent", "Random_Forest_downsampled" = "Random Forest", "Lasso
                              "Gam" = "Gam", "Ensemble" = "Ensemble")) +
  scale_y_continuous(breaks = new_breaks, labels = new_labels) +  # Modify y-axis labels
  # scale_x_discrete(labels = c("Gam","Lasso","RF","Maxent","Ensemble")) +  # Modify y-axis labels
  labs(title = NULL, x = NULL, y = NULL) +  # Removed title and axis labels
  theme_minimal() +
  geom_text(aes(label = round(value, 2),color = ifelse(value > 30, "white", "black")), size = 1.8) +
  scale_color_manual(values = c("black", "white"))+
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 6, face = "bold"),
    axis.text.y = element_text(size = 5, hjust = 3),  # Adjust the size here
    strip.background = element_blank(),
    strip.text = element_text(size = 6,face = "bold"),
    strip.placement = "outside",  # Place the facet label to the left
    legend.position = "none",# Remove the legend
    panel.border = element_rect(color = "black", fill = NA, size = 1),  # Add border around each facet)
    panel.spacing = unit(0.2, "cm"),  # Adjust the space between facets. Change "2" to your desired spa
    axis.ticks.y = element_line(color = "black", size = 0.5),
    axis.ticks.length = unit(0.1, "cm"),  # Adjust the tick length
    axis.ticks.x = element_line(color = "black", size = 0.5))



heattable_100



ggsave(here::here("output/plots/heattable_PA10.jpg"), plot = heattable_same, width = 6, height = 4)
```

Calculate Mean Model Ranks

```r
# Mean Model Ranks ------------------------------------------------------

# function ------------------------------------------------------------------

mean_rank_function <- function(data_ranks) {

  mean_ranks <- data.frame(model=c("GAM", "Lasso", "Maxent", "Random_Forest_downsampled", "Ensemble"),
```

```
                            mean_rank= c(


                              weighted.mean(x = data_ranks[data_ranks$model=="GAM",][[2]],
                                            w=data_ranks$percentage[data_ranks$model=="GAM"]),

                              weighted.mean(x = data_ranks[data_ranks$model=="Lasso",][[2]],
                                            w=data_ranks$percentage[data_ranks$model=="Lasso"]),

                              weighted.mean(x = data_ranks[data_ranks$model=="Maxent",][[2]],
                                            w=data_ranks$percentage[data_ranks$model=="Maxent"]),

                              weighted.mean(x = data_ranks[data_ranks$model=="Random_Forest_downsample
                                            w=data_ranks$percentage[data_ranks$model=="Random_Forest_d



                              weighted.mean(x = data_ranks[data_ranks$model=="Ensemble",][[2]],
                                            w=data_ranks$percentage[data_ranks$model=="Ensemble"])
                            )
                          )

  return(mean_ranks)

}

# calculate ------------------------------------------------------------------


auc_mean_ranks <- mean_rank_function(data_ranks_auc)
boyce_mean_ranks <- mean_rank_function(data_ranks_boyce)
rmse_mean_ranks <- mean_rank_function(data_ranks_rmse)
mae_mean_ranks <- mean_rank_function(data_ranks_MAE)
TSS_mean_ranks <- mean_rank_function(data_ranks_TSS)


# 10 -------------------------------------------------------------------------


auc_mean_ranks_10 <- mean_rank_function(data_ranks_auc_10)
boyce_mean_ranks_10 <- mean_rank_function(data_ranks_boyce_10)
rmse_mean_ranks_10 <- mean_rank_function(data_ranks_rmse_10)
mae_mean_ranks_10 <- mean_rank_function(data_ranks_MAE_10)
TSS_mean_ranks_10 <- mean_rank_function(data_ranks_TSS_10)

# all rank means -------------------------------------------------------------


 # data frame with mean model ranks per algorithim and metric
 all_rank_means <- data.frame(model= boyce_mean_ranks$model,
                              rank_boyce= boyce_mean_ranks$mean_rank,
                              rank_AUC= auc_mean_ranks$mean_rank,
                              rank_MAE= mae_mean_ranks$mean_rank,
```

```
                              rank_rmse= rmse_mean_ranks$mean_rank,
                              rank_TSS= TSS_mean_ranks$mean_rank
                              )

# 10 ------------------------------------------------------------------------


 # data frame with mean model ranks per algorithim and metric
 all_rank_means_10 <- data.frame(model= boyce_mean_ranks_10$model,
                              rank_boyce= boyce_mean_ranks_10$mean_rank,
                              rank_AUC= auc_mean_ranks_10$mean_rank,
                              rank_MAE= mae_mean_ranks_10$mean_rank,
                              rank_rmse= rmse_mean_ranks_10$mean_rank,
                              rank_TSS= TSS_mean_ranks_10$mean_rank
                              )


 # round
 all_ranks_means <- data.frame(all_rank_means$model, round(all_rank_means[,2:6],2))
 all_ranks_means_10 <- data.frame(all_rank_means_10$model, round(all_rank_means_10[,2:6],2))




 # transpose
 all_ranks_means_transpose_10 <- as.data.frame(t(all_ranks_means_10))
 all_ranks_means_transpose_final_10 <- data.frame(all_ranks_means_transpose_10[2:6,])
 names(all_ranks_means_transpose_final_10) <- all_ranks_means_transpose_10[1,]

 all_ranks_means_transpose <- as.data.frame(t(all_ranks_means))
 all_ranks_means_transpose_final <- data.frame(all_ranks_means_transpose[2:6,])
 names(all_ranks_means_transpose_final) <- all_ranks_means_transpose[1,]


 # show
 all_ranks_means_transpose_final
 all_ranks_means_transpose_final_10
```

Compare Performance for multiple threshholds

```
# Comparing Performance for different presence numbers with Continous Boyce Index ---------------------




# setup data
data_boyce <- data.frame(cases=c(rep(10,5),rep(20,5), rep(30,5),rep(100,5)),
                  groups = rep(c("Gam","Lasso","RF","Maxent","Ensemble"),4),
                  mean = c(result_10$Mean$boyceIndex,result_20$Mean$boyceIndex,result_30$Mean$boyceInd
                  sd = c(result_10$se$boyceIndex,result_20$se$boyceIndex,result_30$se$boyceIndex,resul
)
```

```r
# Structure of the data
str(data_boyce)

# Set the levels of the groups factor
data_boyce$groups <- factor(data_boyce$groups, levels = c("Gam", "Lasso", "Maxent","RF", "Ensemble"))

# Color palette
custom_palette <- c("#648FFF", "#DC267F", "#E6340A", "#E8CC10")


# Initialize ggplot with specified data and aesthetics
boyce_plot <- ggplot(data_boyce, aes(x = groups, y = mean, group = cases)) +

  # Plot points colored by 'cases', with size and position adjustments
  geom_point(aes(color = factor(cases)), size = 3, position = position_dodge(0.6)) +

  # Add error bars for standard deviation
  geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd, color = factor(cases)), width = 0.2, position =

  # Set labels for the plot
  labs(
    title = NULL,
    x = NULL,
    y = "Continous Boyce Index",
    color = "Occurrence range"
  ) +

  # Adjust y-axis breaks
  scale_y_continuous(breaks = seq(-0.5, 1, by = 0.2)) +

  # Customize color scale and legend labels
  scale_color_manual(labels = c("> 10", "> 20 ", "> 30", "> 100"), values = custom_palette) +

  # Apply minimal theme and adjust axis/legend properties
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 12, face = "bold"),
        axis.title.y = element_text(vjust = 3, size = 13),
        panel.border = element_blank(),
        axis.line = element_line(color = "black"),
        axis.ticks = element_line(size = 1)) +

  # Adjust legend appearance
  theme(
    legend.text = element_text(size = 12),
    legend.title = element_text(size = 12),
    legend.position = "top",
    legend.key.size = unit(1.5, "cm")
  ) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf, color = "black", fill = NA, size =


  # Modify legend symbol size
  guides(color = guide_legend(override.aes = list(size = 4)))
```

```r
ggsave("output/plots/boyce_threshholds.jpg", plot = boyce_plot, width = 10, height = 7)
```