Muntjacs invading Germany

Potential distribution of the small Deer (Cervidae) species Reeves’ Muntjac (Muntiacus reevesi) in Germany - Code documentation

Louis-Marvin Sander

Table of Contents

knitr::opts\_chunk$set(eval = FALSE)

# Preparation

## Packages

library(raster)  
library(tidyverse)  
library(sf)  
library(terra)  
library(rgdal)  
library(dismo)  
library(tmap)  
library(tmaptools)  
library(magrittr)  
library(pROC)  
library(spdep)  
library(stan4bart)  
library(countrycode)  
library(geodata)  
library(whitebox)  
library(readxl)  
library(ROCR)  
library(scales)  
library(reshape2)  
library(ggbreak)  
library(ggstatsplot)

## Seed

set.seed(2121997)

## Self defined functions

### Negate

`%!in%` <- Negate(`%in%`)

### Get n Minimal/Maximal results

List the first n results of a dataframe sorted by a column. Interesting for seeing the most collinear values late on.

Input: - a = data frame/tibble that sould be displayed - b = column to sort the results - c = number of rows that sould be displayed - mode = c(“max”, “min”) means either sort decreasing or increasing.

owncount <- function(a, b, c = ncol(a), mode = c("max", "min")) {  
 mode <- ifelse(mode == "max", TRUE, FALSE)  
 if (c > nrow(a)) c <- nrow(a)  
 result <- tibble()  
 b <- match(b, colnames(a))  
 result <- a[order(a[, b], decreasing = mode), ] %>% head(., n = c)  
 return(result)  
}

### Data Preparation

It automatically appends the data set with randomly sampled absence points based on the extent of the SpatRaster and creates data frames that can be used in the modelling process The data have to be in the same coordinate reference system.

Attention: Here is implemented directly the rule of thumb after Huberty 1994 for splitting into train and test data.

Inputs: - start = environment as a SpatRaster with different layers as predictors - species = sf object of occurrence points - most.important = vector of variables threatened as important ones that should be extracted from the SpatRaster - randoms = same as most.important but for the random variables

Output: List of: - train.points = sf object of model training points - test.points = sf object of model test points - train.df = data frame of variables at train points + pa = column for presence absence - test.df = data frame of variables at test points + pa = column for presence absence - removedNA = removed points due to NA in an environmental layer

fuzzySim::percentTestData(5) # in our case at the end 5 main variables  
  
extraction\_format <- function(start, species, most.important = NULL, randoms = NULL) {  
 if (class(most.important) != "NULL") {  
 start <- start[[c(most.important, randoms)]]  
 }  
  
 sample <- sample(  
 c(TRUE, FALSE),  
 nrow(species),  
 replace = TRUE,  
 prob = c(0.66, 0.33) # variable split after Huberty  
 )  
  
 species.train.points <- species[sample, ]  
 species.test.points <- species[!sample, ]  
  
 maxabsencepoints <- nrow(species.train.points)  
  
 absences <- dismo::randomPoints(mask = start[[1]] %>% raster(), n = maxabsencepoints, p = species.train.points %>% st\_coordinates())  
 absences.train <- absences  
  
 species.train.df <- data.frame(  
 rbind(  
 terra::extract(start, species.train.points, ID = F),  
 terra::extract(start, absences)  
 ),  
 pa = c(rep(1, nrow(absences)), rep(0, nrow(species.train.points)))  
 ) %>%  
 as\_tibble()  
  
 maxabsencepoints <- nrow(species.test.points)  
  
 absences <- dismo::randomPoints(mask = start[[1]] %>% raster(), n = maxabsencepoints, p = species.test.points %>% st\_coordinates())  
 absences.test <- absences  
  
 species.test.df <- data.frame(  
 rbind(  
 terra::extract(start, species.test.points, ID = F),  
 terra::extract(start, absences)  
 ),  
 pa = c(rep(1, nrow(absences)), rep(0, nrow(species.test.points)))  
 ) %>%  
 as\_tibble()  
  
 train.points <- species.train.points %>%  
 mutate(pa = 1) %>%  
 select(pa, geometry) %>%  
 add\_row(st\_as\_sf(absences.train %>% as\_tibble(), coords = c("x", "y"), crs = st\_crs(species.train.points)) %>% mutate(pa = 0))  
  
 test.points <- species.test.points %>%  
 mutate(pa = 1) %>%  
 select(pa, geometry) %>%  
 add\_row(st\_as\_sf(absences.test %>% as\_tibble(), coords = c("x", "y"), crs = st\_crs(species.test.points)) %>% mutate(pa = 0))  
  
 removedNA <- list(  
 train.data = complete.cases(species.train.df) %>%  
 table() %>%  
 as\_tibble() %>%  
 rename("which" = ".") %>%  
 mutate(which = c("Out", "In")),  
 test.data = complete.cases(species.test.df) %>%  
 table() %>%  
 as\_tibble() %>%  
 rename("which" = ".") %>%  
 mutate(which = c("Out", "In"))  
 )  
 train.points <- train.points[complete.cases(species.train.df), ]  
 test.points <- test.points[complete.cases(species.test.df), ]  
 species.train.df <- species.train.df[complete.cases(species.train.df), ]  
 species.test.df <- species.test.df[complete.cases(species.test.df), ]  
  
 return(  
 list(  
 train.points = train.points,  
 test.points = test.points,  
 train.df = species.train.df,  
 test.df = species.test.df,  
 removedNA = removedNA  
 )  
 )  
}

### Multi collinearity

Adapted “multicol” function from the “fuzzySim” to display more information in the result.

Input: - vars = data frame of environmental conditions - model = not needed in this context - reorder = order result at the end?

Output: - result = tibble with several columns, nrow(result) = ncol(vars) - Rsquared = R² of the built model - Tolerance = 1 - R² - VIF = Variance Inflation Factor 1 / (1 - R²) - count = Number of collinear variables - significant\_vars = Significant collinear variables - insignificant\_vars = Not collinear variables

multicol\_own <- function(vars = NULL, model = NULL, reorder = TRUE) {  
 if (is.null(vars)) {  
 if (is.null(model)) {  
 stop("You must provide either 'vars' or 'model'.")  
 }  
 if (!("glm" %in% class(model))) {  
 stop("'model' must be an object of class 'glm'.")  
 }  
 vars <- model$model[, -1]  
 }  
 vars <- as.data.frame(vars)  
 if (ncol(vars) < 2) {  
 return(message("Cannot compute collinearity with less than two variables."))  
 }  
  
 result <- tibble(factor = colnames(vars))  
  
 for (v in 1:ncol(vars)) {  
 v.name <- colnames(vars)[v]  
 other.v.names <- colnames(vars)[-v]  
 mod.formula <- as.formula(paste(v.name, "~", paste(other.v.names,  
 collapse = "+"  
 )))  
 mod <- lm(mod.formula, data = vars)  
 R2 <- summary(mod)$r.squared  
 result[v, "Rsquared"] <- R2  
 result[v, "Tolerance"] <- 1 - R2  
 result[v, "VIF"] <- 1 / (1 - R2)  
 result[v, "count"] <- (summary(mod)$coefficients[, 4] < 0.05) %>%  
 table() %>%  
 .["TRUE"]  
 result[v, "significant\_vars"] <- enframe(list(names(summary(mod)$coefficients[, 4])[summary(mod)$coefficients[, 4] < 0.05][order(summary(mod)$coefficients[, 1][summary(mod)$coefficients[, 4] < 0.05])]))[2]  
 result[v, "insignificant\_vars"] <- enframe(list(names(summary(mod)$coefficients[, 4])[summary(mod)$coefficients[, 4] > 0.05][order(summary(mod)$coefficients[, 1][summary(mod)$coefficients[, 4] > 0.05])]))[2]  
 }  
  
 if (reorder) {  
 result <- result[order(result$VIF, decreasing = TRUE), ]  
 }  
 return(result)  
}

### Nearest distance

Creation of a vector that represents the distance of a point to the nearest feature. This can be used if there is no raster available, for example if you want to use “distance to streets/urban areas”.

Input: - a = sf object, here presence/absence points - b = sf object, here polygons of cities, lines of streets, … it can also be a named list if more than one variable sould be added

Output: - distance = Distance of every point in *a* to the nearest polygon in *b*

nearestdistance\_to <- function(a, b) {  
 nearest <- vector()  
 distance <- tibble(1:nrow(a))  
  
 for (i in seq(1, length(b), 1)) {  
 nearest <- st\_nearest\_feature(a, b[[i]])  
 distance[, i] <- st\_distance(a, b[[i]][nearest, ], by\_element = TRUE) %>% as.vector()  
 }  
 names(distance) <- names(b)  
  
 return(distance)  
}

### Weights

Creation of weights according to the number of features in a radius, means 0.5/n where n = number of points in a radius. That means, if n = 1 than the occurrence has one neighbor, therefore 0.5/1 = 0.5 and bot occurrences are weighted 0.5. Default: the radius of a circle that results in the average territorial size of the muntjac according to …

Input: - sf\_geom = sf object of points, here muntjac Occurrences - buffer\_radius = radius of a circle that results in the size of an average territory

Output: - vector of length(sf\_geom) with values [1,0[.

nbfunction <- function(sf\_geom, buffer\_radius = 298.5411) {  
 # function from the spdep package, count of neighbours within a radius  
 dnearneigh(sf\_geom, 0, buffer\_radius) %>%  
 # set NA to 0 neigbours  
 lapply(function(x) ifelse(x == 0, NA, length(x))) %>%  
 lapply(unique) %>%  
 unlist() %>%  
 # calculate the weights  
 (function(x) (0.5 / x)) %>%  
 # division by 0 not possible, therefore exchange NA by 0  
 ifelse(is.na(.), 1, .)  
}

### stan4bart raster/terra prediction wrapper

TODO:

Predicts a stan4bart model to a SpatRaster of environmental variables. Only recommended if the raster is relatively small, low number of predictions or predictions to different raster are pending. If you often use the same raster stack for prediction, use the other function where you have to export your raster as table first. Work through found at …!

Input: - model = stan4bart object with “keepTrees = T” - raster = SpatRaster with new environment (for prediction) or old environment (for inter/extrapolation)

Output: - new\_raster = Predicted raster

raster\_pred\_stan4bart <- function(model, raster, type = "ev") {  
 # Raster conversion to df and save places where not NA, afterwards remove NAs  
 data\_map\_df <- as.matrix(raster, xy = F, na.rm = F, byrow = T) %>% as.data.frame()  
 data\_map\_df\_comp <- which(complete.cases(data\_map\_df))  
 data\_map\_df <- data\_map\_df[data\_map\_df\_comp, ]  
  
 # predicting the df with help of the generic function  
 new\_data <- predict.stan4bartFit(  
 object = model,  
 newdata = data\_map\_df,  
 type = type,  
 combine\_chains = TRUE,  
 sample\_new\_levels = TRUE  
 ) %>%  
 rowMeans(na.rm = T) # Here you can tweak how many iterations should be included at the end. If the model needs a long time for reaching convergence it might be interesting to look at different stages. Looking at only one iteration at once, you can see the model "learning".  
  
 # preparation to rasterize the df  
 pre\_raster <- rep(NA, ncell(raster))  
 pre\_raster[data\_map\_df\_comp] <- new\_data  
 pre\_raster %<>%  
 matrix(  
 nrow = nrow(raster),  
 ncol = ncol(raster),  
 byrow = T  
 )  
  
 # create the new raster  
 new\_raster <- rast(  
 pre\_raster,  
 crs = crs(raster),  
 extent = ext(raster)  
 )  
  
 return(new\_raster)  
}

### Brier Score

Get data out of a ROCR-Object and calculate the brier score.

Input: - ROCR-Object

Output: - Brier Score

get\_and\_calc\_bri <- function(x) {  
 tru <- x@labels %>%  
 unlist() %>%  
 as.character() %>%  
 as.numeric()  
 preds <- x@predictions %>% unlist()  
  
 return(  
 (1 / length(tru) \* sum((preds - tru)^2))  
 )  
}

### Efficient prediction

#### Raster preparation

Function that converts a raster into a table for efficient reuse in prediction function.

Input: - in\_raster = Raster stack to be converted - in\_vars = Names of the layers that should be converted - save.path = Path where to store + Base name of the documents. Attention: File ending and identifier are added automatically.

Output = Two output files: - \*\_na-free.txt = Raster as table, every column that does not match “complete.cases()” removed. - \*\_comp\_cases.txt = Vector of line numbers that match “complete.cases()”

raster\_prep\_stan4bart <- function(in\_raster, in\_vars, save.path = tempdir()) {  
 data\_map\_df <- as.matrix(  
 in\_raster[[in\_vars]],  
 xy = F,  
 na.rm = F,  
 byrow = T  
 ) %>%  
 as.data.frame()  
  
 data\_map\_df\_comp <- which(complete.cases(data\_map\_df))  
 data\_map\_df <- data\_map\_df[data\_map\_df\_comp, ]  
  
 write.table(data\_map\_df, paste0(save.path, "\_na-free.txt"))  
  
 write.table(data\_map\_df\_comp, paste0(save.path, "\_comp\_cases.txt"))  
}

#### Fast prediction

TODO: Input output

fast\_predict <- function(model, newdata, complete\_cases, base\_raster, type = "ev", coltake = NULL) {  
 new\_data <- predict(  
 object = model,  
 newdata = newdata,  
 type = type,  
 combine\_chains = TRUE,  
 sample\_new\_levels = TRUE  
 )  
  
 if (is.null(coltake)) {  
 new\_data %<>% rowMeans(na.rm = T)  
 } else {  
 new\_data %<>%  
 .[, coltake] %>%  
 rowMeans(na.rm = T)  
 }  
  
 pre\_raster <- rep(NA, ncell(base\_raster))  
 pre\_raster[complete\_cases$x] <- new\_data  
 pre\_raster %<>%  
 matrix(  
 nrow = nrow(base\_raster),  
 ncol = ncol(base\_raster),  
 byrow = T  
 )  
  
 fast\_result <- rast(  
 pre\_raster,  
 crs = crs(base\_raster),  
 extent = ext(base\_raster)  
 )  
  
 return(fast\_result)  
}  
  
# fast\_predict <- function(model, newdata, complete\_cases, base\_raster, type = "ev", coltake = NULL) {  
# new\_data <- predict(  
# object = model,  
# newdata = newdata,  
# type = type,  
# combine\_chains = TRUE,  
# sample\_new\_levels = TRUE  
# )  
  
  
# return(new\_data)  
# }

#### Finding the threshold

TODO: input output

thresh\_01 <- function(eval\_list, thresh = 0.925) {  
 lapply(eval\_list, function(x) {  
 ROCR::performance(x, measure = "acc")@x.values[[1]][which(ROCR::performance(x, measure = "acc")@y.values[[1]] > 0.925)] %>%  
 unlist() %>%  
 min()  
 }) %>%  
 unlist() %>%  
 mean()  
}

## Directories

dir.create("./1\_input/")  
dir.create("./2\_work")  
dir.create("./3\_ready")  
dir.create("./4\_output")

## Variables

### Boundaries World and Area of interest

#### Choosen Coutries as AOI

countryCodes <- tibble(isoA3 = vector(length = 40))  
countryCodes$isoA3 <- c("AUT", "BEL", "BGR", "HRV", "CYP", "CZE", "DNK", "EST", "FIN", "FRA", "DEU", "GRC", "HUN", "IRL", "ITA", "LVA", "LTU", "LUX", "MLT", "NLD", "POL", "PRT", "ROU", "SVK", "SVN", "ESP", "SWE", "GBR", "TWN", "CHE", "NOR", "BLR", "UKR", "SRB", "MKD", "MNE", "BIH", "HRV", "MDA", "CHN")  
countryCodes$isoA2 <- countrycode(countryCodes$isoA3, "iso3c", "iso2c")

#### World Border

Downloaded from [opendatasoft.com](https://public.opendatasoft.com/explore/dataset/world-administrative-boundaries/information/?location=2,44.59047,46.93359&basemap=jawg.light&dataChart=eyJxdWVyaWVzIjpbeyJjb25maWciOnsiZGF0YXNldCI6IndvcmxkLWFkbWluaXN0cmF0aXZlLWJvdW5kYXJpZXMiLCJvcHRpb25zIjp7fX0sImNoYXJ0cyI6W3siYWxpZ25Nb250aCI6dHJ1ZSwidHlwZSI6ImNvbHVtbiIsImZ1bmMiOiJDT1VOVCIsInNjaWVudGlmaWNEaXNwbGF5Ijp0cnVlLCJjb2xvciI6IiNGRjUxNUEifV0sInhBeGlzIjoic3RhdHVzIiwibWF4cG9pbnRzIjo1MCwic29ydCI6IiJ9XSwidGltZXNjYWxlIjoiIiwiZGlzcGxheUxlZ2VuZCI6dHJ1ZSwiYWxpZ25Nb250aCI6dHJ1ZX0%3D).

world\_border <- st\_read("https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/world-administrative-boundaries/exports/kml?lang=en&timezone=Europe%2FBerlin")  
st\_write(world\_border, "./1\_input/world-administrative-boundaries.shp")  
  
world\_border %<>%  
 select(Name, geometry) %>%  
 st\_transform(crs = 4087)  
  
st\_write(world\_border, "./3\_ready/WORLD\_border.shp")  
world\_border <- st\_read(("./3\_ready/WORLD\_border.shp"))

#### AOI Border

AOI <- world\_border  
  
AOI %<>%  
 select(Name, geometry) %>%  
 filter(Name %in% countryCodes$isoA3)  
  
st\_write(AOI, "./3\_ready/AOI\_border.shp", append = F)  
AOI <- st\_read("./3\_ready/AOI\_border.shp")

### Response (Muntiacus reevesi occurrences)

#### GBIF

Source: [GBIF](https://www.gbif.org/occurrence/download/0065916-230224095556074)

muntgbif <- read\_delim("./1\_input/occurrence.txt", delim = "\t")  
  
muntgbif %<>%  
 select(year, countryCode, decimalLatitude, decimalLongitude) %>%  
 drop\_na() %>%  
 st\_as\_sf(coords = c("decimalLongitude", "decimalLatitude"), crs = 4326) %>%  
 st\_transform(crs = 4087) %>%  
 mutate(source = rep("GBIF", nrow(.))) %>%  
 unique() %>%  
 distinct(geometry, .keep\_all = T) %>%  
 .[AOI, ]

#### Own extraction

muntown <- readxl::read\_xlsx("./1\_input/Own\_muntiac.xlsx")  
  
muntown %<>%  
 st\_as\_sf(coords = c("decimalLongitude", "decimalLatitude"), crs = 4326) %>%  
 st\_transform(crs = 4087) %>%  
 unique() %>%  
 distinct(geometry, .keep\_all = T) %>%  
 mutate(year = as.numeric(year)) %>%  
 .[AOI, ]

#### NBN Atlas

Source: [NBN Atlas](https://ror.org/00mcxye41)

muntnbn <- read\_delim("./1\_input/records-NBN.csv", delim = ",")  
  
muntnbn %<>%  
 select("year processed", "decimalLongitude processed", "decimalLatitude processed") %>%  
 drop\_na() %>%  
 st\_as\_sf(coords = c("decimalLongitude processed", "decimalLatitude processed"), crs = 4326) %>%  
 st\_transform(crs = 4087) %>%  
 mutate(source = rep("NBN", nrow(.)), countryCode = rep("GB", nrow(.))) %>%  
 unique() %>%  
 distinct(geometry, .keep\_all = T) %>%  
 .[AOI, ]  
  
colnames(muntnbn)[1] <- "year"

#### Schleswig Holstein

Source: [Landesamt für Landwirtschaft, Umwelt und ländliche Räume in Schleswig-Holstein](https://www.schleswig-holstein.de/DE/landesregierung/ministerien-behoerden/LLUR/llur_node.html), received from [Dipl.-Geogr. Heiko Schmüser (CAU Kiel)](https://www.landscape-ecology.uni-kiel.de/de/team-1/heiko-schmueser)

muntsw <- st\_read("./1\_input/schmueser muntjac/20221219\_MuntjakDaten\_Multibase.shp")  
  
muntsw %<>%  
 select("Wolfsjahr", "geometry") %>%  
 set\_colnames(c("year", "geometry")) %>%  
 drop\_na() %>%  
 st\_transform(crs = 4087) %>%  
 mutate(source = rep("SW", nrow(.)), countryCode = rep("DE", nrow(.))) %>%  
 unique() %>%  
 distinct(geometry, .keep\_all = T) %>%  
 .[AOI, ]  
  
muntsw$year %<>% gsub(".\*/", "", .) %>% as.numeric()

#### Combine to one dataset

all\_occs <- add\_row(muntgbif, muntnbn) %>%  
 add\_row(muntown) %>%  
 add\_row(muntsw) %>%  
 unique() %>%  
 distinct(geometry, .keep\_all = T)  
  
rm(muntgbif, muntnbn, muntown, muntsw)  
  
st\_write(all\_occs, "./2\_work/all\_occs.shp")

### Water bodies

Sources: [water bodies](https://hub.arcgis.com/content/esri::world-water-bodies/about) and [linear waters](https://www.arcgis.com/home/item.html?id=273980c20bc74f94ac96c7892ec15aff)

Preprocessing via QGIS: - Combine both files - Export as shape with CRS = EPSG4087

water <- st\_read("./2\_work/world\_water.shp")  
  
water %<>%  
 select(ISO\_CC, geometry) %>%  
 sf::st\_simplify(dTolerance = 100)  
  
st\_write(water, "./2\_work/water\_simple.shp")  
water <- st\_read("./2\_work/water\_simple.shp")

#### Check occurrences of ones in waterbodies

water\_dubles <- all\_occs %>% st\_filter(water, .predicate = st\_intersects)  
  
all\_occs %<>% .[all\_occs$geometry %!in% water\_dubles$geometry, ]  
  
st\_write(all\_occs, "./3\_ready/all\_occs.shp", append = F)

### Road data

Source: [Natural Earth](https://naciscdn.org/naturalearth/10m/cultural/ne_10m_roads.zip)

download.file(  
 url = "https://naciscdn.org/naturalearth/10m/cultural/ne\_10m\_roads.zip",  
 destfile = "./1\_input/ne\_10m\_roads.zip"  
)  
  
unzip(  
 "./1\_input/ne\_10m\_roads.zip",  
 exdir = "./1\_input/"  
)  
  
roads <- st\_read("./1\_input/ne\_10m\_roads.shp")  
  
roads %<>%  
 select(type, name, length\_km, continent, geometry) %>%  
 filter(type != "Ferry Route", type != "Ferry, seasonal") %>%  
 st\_transform(crs = 4087)  
  
st\_write(roads, "./3\_ready/roads.shp", append = F)

### Urban areas

Source: [Natural Earth](https://naciscdn.org/naturalearth/10m/cultural/ne_10m_urban_areas.zip)

download.file(  
 url = "https://naciscdn.org/naturalearth/10m/cultural/ne\_10m\_urban\_areas.zip",  
 destfile = "./1\_input/ne\_10m\_urban\_areas.zip"  
)  
  
unzip(  
 "./1\_input/ne\_10m\_urban\_areas.zip",  
 exdir = "./1\_input/"  
)  
  
urban <- st\_read("./1\_input/ne\_10m\_urban\_areas.shp")  
  
urban %<>%  
 select(area\_sqkm, geometry) %>%  
 st\_transform(crs = 4087)  
  
st\_write(urban, "./3\_ready/urban.shp", append = F)

### Elevation + Slope

elevation <- elevation\_global(res = 0.5, path = "./1\_input/elevation")  
  
elevation %<>% terra::project(AOI)  
  
terra::writeRaster(elevation\_work, "./2\_work/elevation.tif", overwrite = T)  
  
slope <- elevation %>%  
 terra::terrain(  
 v = c("slope"),  
 unit = "degrees",  
 filename = "./2\_work/slope.tif",  
 overwrite = T  
 )  
  
elevation <- rast("./2\_work/elevation.tif")  
slope <- rast("./2\_work/slope.tif")

### Climate

climate <- worldclim\_global(  
 var = "bio",  
 res = 0.5,  
 path = "./1\_input/bioclim",  
 version = "2.1"  
)  
  
climate %<>% terra::project(AOI)  
  
terra::writeRaster(climate\_work, "./2\_work/climate.tif", overwrite = T)  
  
climate <- rast("./2\_work/climate.tif")

### Landcover

Get the data and prepare them for stacking with climate.

landcover(var = "trees", path = "./1\_input/landcover")  
landcover(var = "grassland", path = "./1\_input/landcover")  
landcover(var = "shrubs", path = "./1\_input/landcover")  
landcover(var = "cropland", path = "./1\_input/landcover")  
landcover(var = "built", path = "./1\_input/landcover")  
landcover(var = "wetland", path = "./1\_input/landcover")  
  
landCover <- rast(paste0("./1\_input/landcover/", list.files("./1\_input/landcover")))  
  
landCover %<>% terra::project(crs(AOI))  
  
writeRaster(landCover, "./2\_work/landcover.tif")  
  
landCover <- rast("./2\_work/landcover.tif")  
  
mergerast <- rast(ext(climate$wc2.1\_30s\_bio\_1), resolution = res(climate$wc2.1\_30s\_bio\_1), crs = crs(climate), names = names(landCover), nlyrs = nlyr(landCover))  
  
merge(mergerast, landCover, filename = "./2\_work/landcover\_merge.tif")  
landCover <- rast("./2\_work/landcover\_merge.tif")

### Distance maps to Urban areas, Water and streets

#### Step I

The fist step was the rasterization of the urban areas, waters and streets in QGIS. R works too ineficient on large rasters. You can also try the ´fasterize´ function but its not working with MULTILINESTRING.

#### Step II

NAs have to be set to 0 for further processing in whitebox. Additionally the rasters were croped to the AOI.

WORLD\_road\_distance <- rast("./2\_work/WORLD\_roads\_raster.tif")  
WORLD\_road\_distance <- ifel(is.na(WORLD\_road\_distance), 0, WORLD\_road\_distance)  
writeRaster(WORLD\_road\_distance, "./2\_work/WORLD\_roads\_raster\_0\_1.tif", overwrite = T)  
WORLD\_road\_distance %<>% mask(AOI) %>% crop(AOI)  
writeRaster(WORLD\_road\_distance, "./2\_work/AOI\_roads\_raster\_0\_1.tif", overwrite = T)  
  
WORLD\_urban\_distance <- rast("./2\_work/WORLD\_urban\_raster.tif")  
WORLD\_urban\_distance <- ifel(is.na(WORLD\_urban\_distance), 0, WORLD\_urban\_distance)  
writeRaster(WORLD\_urban\_distance, "./2\_work/WORLD\_urban\_raster\_0\_1.tif", overwrite = T)  
WORLD\_urban\_distance %<>% mask(AOI) %>% crop(AOI)  
writeRaster(WORLD\_urban\_distance, "./2\_work/AOI\_urban\_raster\_0\_1.tif", overwrite = T)  
  
WORLD\_water\_distance <- rast("./2\_work/WORLD\_water\_raster.tif")  
WORLD\_water\_distance <- ifel(is.na(WORLD\_water\_distance), 0, WORLD\_water\_distance)  
writeRaster(WORLD\_water\_distance, "./2\_work/WORLD\_water\_raster\_0\_1.tif", overwrite = T)  
WORLD\_water\_distance %<>% mask(AOI) %>% crop(AOI)  
writeRaster(WORLD\_water\_distance, "./2\_work/AOI\_water\_raster\_0\_1.tif", overwrite = T)

#### Step III

Calculade eucledian distance maps via whitebox.

wbt\_euclidean\_distance(  
 "./2\_work/AOI\_roads\_raster\_0\_1.tif",  
 "./2\_work/AOI\_road\_distance.tif"  
)  
  
wbt\_euclidean\_distance(  
 "./2\_work/AOI\_urban\_raster\_0\_1.tif",  
 "./2\_work/AOI\_urban\_distance.tif"  
)  
  
wbt\_euclidean\_distance(  
 "./2\_work/AOI\_water\_raster\_0\_1.tif",  
 "./2\_work/AOI\_water\_distance.tif"  
)

#### Combine all + Crop to AOI + Resample for faster prelimnary model runs

The distane maps are not available in world size, therefore the distances to roads, waters and urban areas have to be extracted on an other way lateron. That means that the AOI\_environment.tif is fully self contained in all environmental factors that are needed but the WORLD\_environment.tif is not.

TODO: umbenennen der variablen ohne dieses wc\_2.1 krams

WORLD\_environment <- terra::rast(  
 c(  
 "./2\_work/elevation.tif",  
 "./2\_work/slope.tif",  
 "./2\_work/climate.tif",  
 "./2\_work/landcover\_merge.tif"  
 )  
)  
  
names(WORLD\_environment) %<>% gsub(".\*s\_", "", .)  
  
writeRaster(WORLD\_environment, "./3\_ready/WORLD\_environment.tif")  
  
WORLD\_environment <- rast("./3\_ready/WORLD\_environment.tif")  
  
AOI\_environment <- WORLD\_environment %>%  
 crop(AOI) %>%  
 mask(AOI) %>%  
 c(  
 .,  
 rast(c(  
 "./2\_work/AOI\_road\_distance.tif",  
 "./2\_work/AOI\_urban\_distance.tif",  
 "./2\_work/AOI\_water\_distance.tif"  
 ))  
 )  
  
names(AOI\_environment) %<>% gsub("AOI\_", "", .)  
  
writeRaster(AOI\_environment, "./3\_ready/AOI\_environment.tif")  
  
AOI\_environment <- rast("./3\_ready/AOI\_environment.tif")  
  
EU\_environment <- rast("./3\_ready/AOI\_environment.tif") %>%  
 crop(AOI[AOI$Name %!in% c("TWN", "CHN"), ]) %>%  
 terra::mask(AOI[AOI$Name %!in% c("TWN", "CHN"), ])  
  
writeRaster(EU\_environment, "./3\_ready/EU\_environment.tif")  
  
EU\_environment <- rast("./3\_ready/EU\_environment.tif")  
  
# Resample with stacks is not possible at this moment in whitebox but will hopefully be included in future, therefore use the terra function, velox or QGIS.  
  
wbt\_resample(  
 inputs = "./3\_ready/WORLD\_environment.tif",  
 output = "./3\_ready/WORLD\_environment\_low.tif",  
 cell\_size = 5000,  
 method = "cc"  
)  
  
wbt\_resample(  
 inputs = "./3\_ready/AOI\_environment.tif",  
 output = "./3\_ready/AOI\_environment\_low.tif",  
 cell\_size = 5000,  
 method = "cc"  
)  
  
wbt\_resample(  
 inputs = "./3\_ready/EU\_environment.tif",  
 output = "./3\_ready/EU\_environment\_low.tif",  
 cell\_size = 5000,  
 method = "cc"  
)  
  
# Terra aggregrate  
  
WORLD\_environment\_low <- terra::aggregate(WORLD\_environment, fact = 10, filename = "./3\_ready/WORLD\_environment\_low.tif")  
AOI\_environment\_low <- terra::aggregate(AOI\_environment, fact = 10, filename = "./3\_ready/AOI\_environment\_low.tif")  
EU\_environment\_low <- terra::aggregate(EU\_environment, fact = 10, filename = "./3\_ready/EU\_environment\_low.tif")  
  
WORLD\_environment\_low <- rast("./3\_ready/WORLD\_environment\_low.tif")  
AOI\_environment\_low <- rast("./3\_ready/AOI\_environment\_low.tif")  
EU\_environment\_low <- rast("./3\_ready/EU\_environment\_low.tif")

### Future conditions

#### Future climate

Source: [World Climate Reseach Programme](https://doi.org/10.22033/ESGF/CMIP6.4332)

Can be cropped directly because of no need of data outside of AOI for prediction.

future\_climate <- rast("./1\_input/future data/wc2.1\_2.5m\_bioc\_ACCESS-CM2\_ssp585\_2061-2080.tif")[[c(1, 12)]]  
  
future\_climate %<>% terra::project(AOI) %>%  
 crop(AOI[AOI$Name %!in% c("TWN", "CHN"), ]) %>%  
 terra::mask(AOI[AOI$Name %!in% c("TWN", "CHN"), ])  
  
names(future\_climate) <- c("bio\_1", "bio\_12")  
  
dir.create("./2\_work/future data")  
  
writeRaster(future\_climate, "./2\_work/future data/EU\_climate.tif", overwrite = T)

#### Future Urban

Source: [Land-Use Harmonization²](https://luh.umd.edu/data.shtml) and [Extractions made available here](https://doi.org/10.17161/bi.v16i1.15483)

For prediction only AOI is needed.

future\_urban <- rast("./1\_input/future data/CMIP6\_Land\_Use\_Harmonization\_urban\_SSP5\_85\_2071.tif")  
  
future\_urban %<>%  
 terra::project(AOI) %>%  
 crop(AOI[AOI$Name %!in% c("TWN", "CHN"), ]) %>%  
 terra::mask(AOI[AOI$Name %!in% c("TWN", "CHN"), ])  
  
writeRaster(future\_urban, "./2\_work/future data/urban\_low.tif", overwrite = T)

#### Future Forest

Source: [Land-Use Harmonization²](https://luh.umd.edu/data.shtml) and [Extractions made available here](https://doi.org/10.17161/bi.v16i1.15483)

Here two forest raster files: primary and secondary forest.

future\_forest <- rast(c("./1\_input/future data/CMIP6\_Land\_Use\_Harmonization\_primf\_SSP5\_85\_2071.tif", "./1\_input/future data/CMIP6\_Land\_Use\_Harmonization\_secdf\_SSP5\_85\_2071.tif"))  
  
future\_forest <- future\_forest[[1]] + future\_forest[[2]]  
  
future\_forest %<>% terra::project(AOI) %>%  
 crop(AOI[AOI$Name %!in% c("TWN", "CHN"), ]) %>%  
 terra::mask(AOI[AOI$Name %!in% c("TWN", "CHN"), ])  
  
writeRaster(future\_forest, "./2\_work/future data/forest\_low.tif", overwrite = T)

#### Resampling

Attention: In my case, teh whitebox had a bug in piping the resample function to the shell, therefore I performed these steps in the GUI (whitebox\_runner).

##### Bug in the resampling function fixed

I dont know when it is fixed, therefore I included the fixed function:

wbt\_resample <- function(  
 inputs, output, cell\_size = NULL, base = NULL, method = "cc",  
 wd = NULL, verbose\_mode = FALSE, compress\_rasters = FALSE,  
 command\_only = FALSE) {  
 wbt\_init()  
 args <- ""  
 args <- paste(args, paste0("--inputs=", whitebox:::wbt\_file\_path(inputs)))  
 args <- paste(args, paste0("--output=", whitebox:::wbt\_file\_path(output)))  
 if (!is.null(cell\_size)) {  
 args <- paste(args, paste0("--cell\_size=", cell\_size))  
 }  
 if (!is.null(base)) {  
 args <- paste(args, paste0("--base=", whitebox:::wbt\_file\_path(base)))  
 }  
 if (!is.null(method)) {  
 args <- paste(args, paste0("--method=", method))  
 }  
 if (!missing(wd)) {  
 args <- paste(args, paste0("--wd=", whitebox:::wbt\_file\_path(wd)))  
 }  
 if (!missing(compress\_rasters)) {  
 args <- paste(args, paste0("--compress\_rasters=", compress\_rasters))  
 }  
 tool\_name <- "resample"  
 wbt\_run\_tool(tool\_name, args, verbose\_mode, command\_only)  
}

##### Resampling

wbt\_resample(  
 inputs = "./2\_work/future data/forest\_low.tif",  
 output = "./2\_work/future data/forest\_high.tif",  
 base = "./2\_work/future data/EU\_climate.tif",  
 method = "cc"  
)  
  
wbt\_resample(  
 inputs = "./2\_work/future data/urban\_low.tif",  
 output = "./2\_work/future data/urban\_high.tif",  
 base = "./2\_work/future data/EU\_climate.tif",  
 method = "cc"  
)  
  
future\_EU\_environment <- rast("./3\_ready/EU\_environment.tif")[[c("slope", "water\_distance", "road\_distance")]]  
  
writeRaster(future\_EU\_environment[[1]], "./2\_work/future data/EU\_environment\_slope.tif")  
writeRaster(future\_EU\_environment[[2]], "./2\_work/future data/EU\_environment\_water\_dist.tif")  
writeRaster(future\_EU\_environment[[3]], "./2\_work/future data/EU\_environment\_road\_dist.tif")  
  
wbt\_resample(  
 inputs = "./2\_work/future data/EU\_environment\_slope.tif",  
 output = "./2\_work/future data/EU\_environment\_slope\_final.tif",  
 base = "./2\_work/future data/EU\_climate.tif",  
 method = "cc"  
)  
  
wbt\_resample(  
 inputs = "./2\_work/future data/EU\_environment\_water\_dist.tif",  
 output = "./2\_work/future data/EU\_environment\_water\_dist\_final.tif",  
 base = "./2\_work/future data/EU\_climate.tif",  
 method = "cc"  
)  
  
wbt\_resample(  
 inputs = "./2\_work/future data/EU\_environment\_road\_dist.tif",  
 output = "./2\_work/future data/EU\_environment\_road\_dist\_final.tif",  
 base = "./2\_work/future data/EU\_climate.tif",  
 method = "cc"  
)

#### New urban distance map

future\_urban <- rast("./2\_work/future data/urban\_high.tif")  
future\_urban <- ifel(is.na(future\_urban) | future\_urban < 0.1, 0, 1)  
writeRaster(future\_urban, "./2\_work/future data/urban\_distance.tif", overwrite = T)  
future\_urban %<>% mask(AOI) %>% crop(AOI)  
writeRaster(future\_urban, "./2\_work/future data/urban\_distance.tif", overwrite = T)  
  
wbt\_euclidean\_distance(  
 "./2\_work/future data/urban\_distance.tif",  
 "./2\_work/future data/urban\_distance.tif"  
)

#### Combine

future\_final <- rast(  
 c(  
 "./2\_work/future data/EU\_environment\_slope\_final.tif",  
 "./2\_work/future data/EU\_environment\_water\_dist\_final.tif",  
 "./2\_work/future data/EU\_environment\_road\_dist\_final.tif",  
 "./2\_work/future data/EU\_climate.tif",  
 "./2\_work/future data/forest\_high.tif",  
 "./2\_work/future data/urban\_distance.tif"  
 )  
)  
  
names(future\_final) <- c("slope", "water\_distance", "road\_distance", "bio\_1", "bio\_12", "trees", "urban\_distance")  
  
writeRaster(future\_final, "./3\_ready/FUTURE\_EU\_environment.tif")  
  
future\_final <- rast("./3\_ready/FUTURE\_EU\_environment.tif")

# Data extraction

dir.create("./3\_ready/model input")

## Input environment

### World high resolution

all\_data <- extraction\_format(  
 start = WORLD\_environment,  
 species = all\_occs  
)  
  
all\_data$train.df %<>%  
 add\_column(  
 nearestdistance\_to(  
 all\_data$train.points,  
 list(  
 road\_distance = roads,  
 urban\_distance = urban,  
 water\_distance = water  
 )  
 )  
 )  
  
all\_data$test.df %<>%  
 add\_column(  
 nearestdistance\_to(  
 all\_data$test.points,  
 list(  
 road\_distance = roads,  
 urban\_distance = urban,  
 water\_distance = water  
 )  
 )  
 )  
  
save(all\_data, file = "./3\_ready/model input/PA\_WORLD.Rdata")  
  
load("./3\_ready/model input/PA\_WORLD.Rdata")

Now jump to the step “Collinearity”, afterwards, when determine the least collinear variables, proceed here.

TODO: AOI and WORLD low resolution not necessary

### World low resolution

all\_data <- extraction\_format(  
 start = WORLD\_environment\_low,  
 species = all\_occs  
)  
  
all\_data$train.df %<>%  
 add\_column(  
 nearestdistance\_to(  
 all\_data$train.points,  
 list(  
 road\_distance = roads,  
 urban\_distance = urban,  
 water\_distance = water  
 )  
 )  
 )  
  
all\_data$test.df %<>%  
 add\_column(  
 nearestdistance\_to(  
 all\_data$test.points,  
 list(  
 road\_distance = roads,  
 urban\_distance = urban,  
 water\_distance = water  
 )  
 )  
 )  
  
save(all\_data, file = "./3\_ready/model input/PA\_WORLD\_low.Rdata")  
  
load("./3\_ready/model input/PA\_WORLD\_low.Rdata")

### AOI high resolution

all\_data <- extraction\_format(  
 start = AOI\_environment,  
 species = all\_occs  
)  
  
save(all\_data, file = "./3\_ready/model input/PA\_AOI.Rdata")  
  
load("./3\_ready/model input/PA\_AOI.Rdata")

### AOI low resolution

all\_data <- extraction\_format(  
 start = AOI\_environment\_low,  
 species = all\_occs  
)  
  
save(all\_data, file = "./3\_ready/model input/PA\_AOI\_low.Rdata")  
  
load("./3\_ready/model input/PA\_AOI\_low.Rdata")

## Prediction environment

Define the choosen predictors and choosen randoms from the step “Collinearity”.

most.important <- c("bio\_1", "bio\_12", "trees", "water\_distance", "slope")  
randoms <- c("road\_distance", "urban\_distance")

Export raster environment as table for faster repetitive prediction.

raster\_prep\_stan4bart(  
 rast("./3\_ready/EU\_environment.tif"),  
 c(most.important, randoms),  
 "./3\_ready/model input/EU\_environment\_rast"  
)  
  
raster\_prep\_stan4bart(  
 rast("./3\_ready/EU\_environment\_low.tif"),  
 c(most.important, randoms),  
 "./3\_ready/model input/EU\_environment\_low\_rast"  
)  
  
raster\_prep\_stan4bart(  
 rast("./3\_ready/FUTURE\_EU\_environment.tif"),  
 c(most.important, randoms),  
 "./3\_ready/model input/FUTURE\_environment\_rast"  
)

# Collinearity

## Test

load("./3\_ready/model input/PA\_WORLD.Rdata")  
  
col\_test\_df <- all\_data$train.df %>%  
 add\_row(all\_data$test.df) %>%  
 filter(pa == 1)  
  
col\_result <- col\_test\_df %>%  
 select(bio\_1, bio\_12, road\_distance, trees, shrubs, cropland, water\_distance, slope, grassland, urban\_distance, water) %>%  
 multicol\_own()  
  
owncount(col\_result, "VIF", 10, "max")  
  
col\_result <- col\_test\_df %>%  
 select(bio\_1, bio\_12, trees, water\_distance, slope) %>%  
 multicol\_own()  
  
owncount(col\_result, "VIF", 10, "max")

TODO: - Results can be found in the publication in table …

Remaining variables: - Annual Precipitation (bio\_12) - Slope (slope) - Annual Mean Temperature (bio\_1) - Tree cover in % (trees) - Water distance in m (water\_distance)

For expected sampling bias the following were included as random factors: - Road distance (road\_distance) - Urban distance (urban\_distance)

# Model

## Output Directories

dir.create("./4\_output/1.EU\_high\_res\_CV\_PRED\_low\_res/")  
dir.create("./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/")  
dir.create("./4\_output/3.WORLD\_high\_res\_CV\_PRED\_low\_res/")  
dir.create("./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/")  
dir.create("./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/")  
dir.create("./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/")

## Model + Predictions

# Settings for the test runs:  
cv.runs <- 5  
iter\_own <- 30  
warmup\_own <- 10  
chains\_own <- 2  
  
  
# Settings for the final run:  
# cv.runs <- 25  
# iter\_own <- 500  
# warmup\_own <- 200  
# chains\_own <- 3  
  
### In the following:  
## Comment or uncomment the necessary parts for predicting AOI or WORLD data sets to high/low resolution EU or the FUTURE data set.  
  
# ################ 1  
# Predict from AOI to low EU  
load("./3\_ready/model input/PA\_AOI.Rdata")  
path\_to\_store <- "./4\_output/1.EU\_high\_res\_CV\_PRED\_low\_res/"  
prediction\_to\_what <- rast("./3\_ready/EU\_environment\_low.tif")  
data\_map\_df <- read.table("./3\_ready/model input/EU\_environment\_low\_rast\_na-free.txt")  
data\_map\_df\_comp <- read.table("./3\_ready/model input/EU\_environment\_low\_rast\_comp\_cases.txt")  
  
  
# ################ 2  
# Predict from AOI to high EU  
# load("./3\_ready/model input/PA\_AOI.Rdata")  
# path\_to\_store <- "./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/"  
# prediction\_to\_what <- rast("./3\_ready/EU\_environment.tif")  
# data\_map\_df <- read.table("./3\_ready/model input/EU\_environment\_rast\_na-free.txt")  
# data\_map\_df\_comp <- read.table("./3\_ready/model input/EU\_environment\_rast\_comp\_cases.txt")  
  
# ################ 3  
# Predict from WORLD to low EU  
# load("./3\_ready/model input/PA\_WORLD.Rdata")  
# path\_to\_store <- "./4\_output/3.WORLD\_high\_res\_CV\_PRED\_low\_res/"  
# prediction\_to\_what <- rast("./3\_ready/EU\_environment\_low.tif")  
# data\_map\_df <- read.table("./3\_ready/model input/EU\_environment\_low\_rast\_na-free.txt")  
# data\_map\_df\_comp <- read.table("./3\_ready/model input/EU\_environment\_low\_rast\_comp\_cases.txt")  
  
# ################ 4  
# Predict from WORLD to high EU  
# load("./3\_ready/model input/PA\_WORLD.Rdata")  
# path\_to\_store <- "./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/"  
# prediction\_to\_what <- rast("./3\_ready/EU\_environment.tif")  
# data\_map\_df <- read.table("./3\_ready/model input/EU\_environment\_rast\_na-free.txt")  
# data\_map\_df\_comp <- read.table("./3\_ready/model input/EU\_environment\_rast\_comp\_cases.txt")  
  
################## 5  
# Predict from AOI to FUTURE  
# load("./3\_ready/model input/PA\_AOI.Rdata")  
# path\_to\_store <- "./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/"  
# prediction\_to\_what <- rast("./3\_ready/FUTURE\_EU\_environment.tif")  
# data\_map\_df <- read.table("./3\_ready/model input/FUTURE\_environment\_rast\_na-free.txt")  
# data\_map\_df\_comp <- read.table("./3\_ready/model input/FUTURE\_environment\_rast\_comp\_cases.txt")  
  
################## 6  
# Predict from WORLD to FUTURE  
# load("./3\_ready/model input/PA\_WORLD.Rdata")  
# path\_to\_store <- "./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/"  
# prediction\_to\_what <- rast("./3\_ready/FUTURE\_EU\_environment.tif")  
# data\_map\_df <- read.table("./3\_ready/model input/FUTURE\_environment\_rast\_na-free.txt")  
# data\_map\_df\_comp <- read.table("./3\_ready/model input/FUTURE\_environment\_rast\_comp\_cases.txt")  
  
  
  
  
# Do not change!  
prediction.count.cv <- 0 # Has to be 0  
  
# Empty output vectors  
model.stan.cv <- list()  
model.stan.predict.cv <- list()  
PA\_test\_pred.cv <- list()  
evaluation.cv <- list()  
auc.cv <- list()  
tss.thresh <- list()  
tss <- list()  
bri <- list()  
accurr <- list()  
acceptance <- list()  
conf.matrix <- list()  
convergence <- list()  
tree\_vars\_imp <- list()  
  
# Preparing data for CV  
Presences.cv <- all\_data$train.df %>%  
 filter(pa == 1) %>%  
 add\_row(  
 all\_data$test.df %>%  
 filter(pa == 1)  
 )  
  
Absences.cv <- all\_data$train.df %>%  
 filter(pa == 0) %>%  
 add\_row(  
 all\_data$test.df %>%  
 filter(pa == 0)  
 )  
  
all.points <- all\_data$train.points %>%  
 filter(pa == 1) %>%  
 add\_row(all\_data$test.points %>% filter(pa == 1)) %>%  
 add\_row(all\_data$train.points %>% filter(pa == 0)) %>%  
 add\_row(all\_data$test.points %>% filter(pa == 0))  
  
  
#################### Automatic CV loop  
  
for (i in 1:cv.runs) {  
 # Random sampling  
 sample.pres <- sample(  
 c(TRUE, FALSE),  
 nrow(Presences.cv),  
 replace = TRUE,  
 prob = c(0.67, 0.33)  
 )  
  
 sample.abs <- sample(  
 c(TRUE, FALSE),  
 nrow(Absences.cv),  
 replace = TRUE,  
 prob = c(0.67, 0.33)  
 )  
  
 PA\_train.cv <- Presences.cv[sample.pres, ] %>% rbind(Absences.cv[sample.abs, ])  
 PA\_train.cv %<>% .[, c(most.important, randoms, "pa")]  
 PA\_test.cv <- Presences.cv[!sample.pres, ] %>% rbind(Absences.cv[!sample.abs, ])  
 PA\_test.cv %<>% .[, c(most.important, randoms, "pa")]  
  
 # Creating weights  
 my\_weights.cv <- nbfunction(all.points$geometry[c(sample.pres, sample.abs)], buffer\_radius = 298.5411)  
 print(paste("Weights", i, "created!"))  
  
 # Defining and running the model  
 model.stan.cv[[i]] <- stan4bart(  
 formula = pa ~ bart(. - urban\_distance - road\_distance) +  
 (1 + road\_distance) + (1 + urban\_distance),  
 verbose = 1, # print progress  
 data = PA\_train.cv, # train data  
 cores = 3, # Cores to use  
 weights = my\_weights.cv, # Weights to use  
 chains = chains\_own, # No. of independent created chains  
 iter = iter\_own, # No. of iterations and [result = iter - warmup]  
 warmup = warmup\_own, # No. of "iter" to draw and reject for warmup  
 bart\_args = list(  
 weights = my\_weights.cv, # Weights to use  
 n.trees = 50, # No. of trees to form  
 keepTrees = T, # Save trees for prediction  
 combineChains = F, # Combine seperate chains to one  
 n.chains = 1 # Nr. of MCMC-Chains within the bart component  
 ),  
 )  
  
 print(paste("Model", i, "created!"))  
  
 # Variable importance  
 tree\_vars\_imp[[i]] <- apply(model.stan.cv[[i]]$bart\_varcount, 3, FUN = rowMeans) %>%  
 rowMeans() %>%  
 as.data.frame() %>%  
 cbind((apply(model.stan.cv[[i]]$bart\_varcount, c(1, 3), FUN = sd) %>%  
 rowMeans() %>%  
 as.data.frame())) %>%  
 setNames(c("mean", "sd")) %>%  
 divide\_by(sum(.$mean))  
  
 # inclusion of coltake  
 # tree\_vars\_imp[[i]] <- model.stan.cv[[i]]$bart\_varcount %>% .[,coltake[1],] %>%  
 # rowMeans() %>%  
 # as.data.frame() %>%  
 # cbind((apply(model.stan.cv[[i]]$bart\_varcount %>% .[,coltake[1],], 1, FUN = sd) %>%  
 # as.data.frame())) %>%  
 # setNames(c("mean", "sd")) %>%  
 # divide\_by(sum(.$mean))  
  
 # Definition of how many iterations and chains should be combined for prediction (needs to be activated in the prediction for testing and further down for the raster)  
 # coltake <- seq(iter\_own - warmup\_own, (iter\_own - warmup\_own) \* chains\_own, iter\_own - warmup\_own)  
  
 # Prediction of the model  
 testpred <- predict(model.stan.cv[[i]], newdata = PA\_test.cv) %>%  
 # .[, coltake] %>% # uncomment if coltake should be used  
 rowMeans()  
  
 # Create a ROCR-object from the prediction + calculate several indices  
 evaluation.cv[[i]] <- ROCR::prediction(testpred, PA\_test.cv$pa)  
  
 auc.cv[[i]] <- ROCR::performance(evaluation.cv[[i]], measure = "auc")@y.values %>%  
 unlist() # area under the curve  
 tss[[i]] <- max(  
 ROCR::performance(evaluation.cv[[i]], measure = "tpr")@y.values %>%  
 unlist() +  
 ROCR::performance(evaluation.cv[[i]], measure = "tnr")@y.values %>%  
 unlist() - 1  
 ) # true skill statistics  
 bri[[i]] <- get\_and\_calc\_bri(evaluation.cv[[i]]) # brier score  
 tss.thresh[[i]] <- ROCR::performance(evaluation.cv[[i]], measure = "acc")@x.values[[1]][which(ROCR::performance(evaluation.cv[[i]], measure = "acc")@y.values[[1]] == max(ROCR::performance(evaluation.cv[[i]], measure = "acc")@y.values[[1]]))] %>%  
 unlist() %>%  
 min() # "best" threshold at maximum accurracy  
 accurr[[i]] <- max(ROCR::performance(evaluation.cv[[i]], measure = "acc")@y.values[[1]]) # accurracy  
 acceptance[[i]] <- ifelse(accurr[[i]] > 0.9 & auc.cv[[i]] > 0.9 & bri[[i]] < 0.05, TRUE, FALSE) # can the model be accepted?  
 convergence[[i]] <- model.stan.cv[[i]]$stan[6, , ] # internal convergence test  
  
 # prediction if model is accepted  
 if (acceptance[[i]]) {  
 print(paste("Model", i, "accepted!"))  
 prediction.count.cv <- prediction.count.cv + 1  
  
 # Predict the model on basis of the new data (extracted rasters from before) and build new rasters  
 model.stan.predict.cv <- fast\_predict(  
 model = model.stan.cv[[i]],  
 newdata = data\_map\_df,  
 complete\_cases = data\_map\_df\_comp,  
 base\_raster = prediction\_to\_what,  
 type = "ev",  
 coltake = NULL # exchange NULL to coltake if you want to use the coltake columns  
 )  
  
 names(model.stan.predict.cv) <- paste("Run", i)  
  
 # Save created raster  
 writeRaster(model.stan.predict.cv,  
 paste0(path\_to\_store, "Run\_", i, ".tif"),  
 overwrite = T  
 )  
 print(paste("Model", i, "saved!"))  
  
 # remove prediction from RAM  
 rm(model.stan.predict.cv)  
 } else {  
 print(paste("Model", i, "rejected!"))  
 }  
  
 # save model and test results in case of PC crashing when calculating big models  
 if (i %% 2 == 0) {  
 save(evaluation.cv, auc.cv, tss.thresh, tss, acceptance, accurr, convergence, tree\_vars\_imp, bri, file = paste0(path\_to\_store, "test\_stat.RData"))  
 }  
} # end of the loop  
  
# save last state data  
save(evaluation.cv, auc.cv, tss.thresh, tss, acceptance, accurr, convergence, tree\_vars\_imp, bri, file = paste0(path\_to\_store, "test\_stat.RData"))  
  
# load all created predictions  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# Calculate Mean and standard deviation  
model.stan.predict.cv <- c(model.stan.predict.cv, model.stan.predict.cv %>% mean(na.rm = T))  
model.stan.predict.cv <- c(model.stan.predict.cv, model.stan.predict.cv[[-(nlyr(model.stan.predict.cv))]] %>% stdev(na.rm = T))  
  
# save mean  
writeRaster(model.stan.predict.cv[["mean"]],  
 paste0(path\_to\_store, "Run\_mean.tif"),  
 overwrite = T  
)  
  
# save standard deviation  
writeRaster(model.stan.predict.cv[["std"]],  
 paste0(path\_to\_store, "Run\_std.tif"),  
 overwrite = T  
)  
print("############### DONE! ###############")

# Exploring the data

## Load right data for further proceeding

# ################ 1  
# Predict from AOI to low EU  
path\_to\_store <- "./4\_output/1.EU\_high\_res\_CV\_PRED\_low\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# ################ 2  
# Predict from AOI to high EU  
path\_to\_store <- "./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# ################ 3  
# Predict from WORLD to low EU  
path\_to\_store <- "./4\_output/3.WORLD\_high\_res\_CV\_PRED\_low\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# ################ 4  
# Predict from WORLD to high EU  
path\_to\_store <- "./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# ################ 5  
# Predict from AOI to FUTURE  
path\_to\_store <- "./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))  
  
# ################ 6  
# Predict from WORLD to FUTURE  
path\_to\_store <- "./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
model.stan.predict.cv <- rast(paste0(path\_to\_store, list.files(path\_to\_store, pattern = "\*.tif")))

## Variable importance

tree\_vars\_imp\_mean <- array(unlist(tree\_vars\_imp), dim = c(nrow(tree\_vars\_imp[[1]]), 2, length(tree\_vars\_imp))) %>%  
 apply(1:2, mean) %>%  
 as\_tibble() %>%  
 reframe(mean = V1, sd = V2)  
  
# Mean Importance over all chains and iterations  
ggplot(tree\_vars\_imp\_mean \* 100, aes(y = rownames(tree\_vars\_imp[[1]]), x = mean)) +  
 geom\_point() +  
 geom\_pointrange(aes(xmin = mean - sd, xmax = mean + sd)) +  
 xlab("Permutation importance (percentage in which the predictor is responsible for a new branch) + mean SD across the chains and CV-runs") +  
 ylab("Predictors in BART component")  
  
  
# Convert matrix to data frame for use in ggplot2  
my\_dataframe <- reshape2::melt(tree\_vars\_imp, level = 2) %>% mutate(name = rep(rownames(tree\_vars\_imp[[1]]), max(L2) \* 2))  
  
# Create the plot with facets  
# X-Axis = Iterations  
# Y-Axis = Variable importance  
# Boxplot = Importance of that variable over the iterations  
# Line = Exact variable importance per iteration  
ggplot(my\_dataframe %>% filter(variable == "mean"), aes(x = L2, y = value, group = name)) +  
 geom\_line(aes(col = name)) +  
 geom\_boxplot(aes(col = name), fill = NA)

## Test statistics

evaluation.cv # Can be used in the ROCR internal statistic functions  
  
# pre evaluated statistics  
auc.cv  
tss.thresh  
tss  
acceptance  
accurr  
convergence  
bri

## Result map stacks

### Directory

dir.create("./4\_output/Result")

### Today

#### Mean over all CV-Runs and base datasets

Curr\_EU\_mean <- rast("./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/Run\_mean.tif")  
  
path\_to\_store <- "./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
  
thresh1 <- thresh\_01(evaluation.cv)  
  
Curr\_EU\_mean[["thresh"]] <- ifel(Curr\_EU\_mean > thresh1, 1, 0)  
  
####### World  
  
  
Curr\_WORLD\_mean <- rast("./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/Run\_mean.tif")  
  
path\_to\_store <- "./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
  
thresh1 <- thresh\_01(evaluation.cv)  
  
Curr\_WORLD\_mean[["thresh"]] <- ifel(Curr\_WORLD\_mean > thresh1, 1, 0)  
  
final\_map <- c(Curr\_WORLD\_mean[["thresh"]], Curr\_EU\_mean[["thresh"]]) %>% mean()  
  
final\_map <- ifel(final\_map == 1, 2, final\_map)  
final\_map <- ifel(final\_map == 0.5, 1, final\_map)  
  
cls <- data.frame(id = seq(0, 2, 1), cover = c("Not Suitable", "1/2 Agrees", "Both Agree"))  
levels(final\_map) <- cls

#### Standard deviation

Curr\_ALL\_std <- rast(c(  
 "./4\_output/2.EU\_high\_res\_CV\_PRED\_high\_res/Run\_std.tif",  
 "./4\_output/4.WORLD\_high\_res\_CV\_PRED\_high\_res/Run\_std.tif"  
)) %>% mean()

#### Final stack

final\_res <- c(Curr\_EU\_mean, Curr\_WORLD\_mean, final\_map, Curr\_ALL\_std)  
  
names(final\_res) <- c("EU\_mean", "EU\_thresh", "WORLD\_mean", "WORLD\_thresh", "final", "std")  
  
terra::writeRaster(final\_res, "./4\_output/Result/TODAY\_Final\_map.tif", overwrite = T)

### Future

#### Mean over all CV-Runs and base datasets

Fut\_EU\_mean <- rast("./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/Run\_mean.tif")  
  
path\_to\_store <- "./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
  
thresh1 <- thresh\_01(evaluation.cv)  
  
Fut\_EU\_mean[["thresh"]] <- ifel(Fut\_EU\_mean > thresh1, 1, 0)  
  
####### World  
  
Fut\_WORLD\_mean <- rast("./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/Run\_mean.tif")  
  
path\_to\_store <- "./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/"  
load(paste0(path\_to\_store, "test\_stat.RData"))  
  
thresh1 <- thresh\_01(evaluation.cv)  
  
Fut\_WORLD\_mean[["thresh"]] <- ifel(Fut\_WORLD\_mean > thresh1, 1, 0)  
  
final\_map <- c(Fut\_WORLD\_mean[["thresh"]], Fut\_EU\_mean[["thresh"]]) %>% mean()  
  
final\_map <- ifel(final\_map == 1, 2, final\_map)  
final\_map <- ifel(final\_map == 0.5, 1, final\_map)  
  
cls <- data.frame(id = seq(0, 2, 1), cover = c("Not Suitable", "1/2 Agrees", "Both Agree"))  
levels(final\_map) <- cls

#### Standard deviation

Fut\_ALL\_std <- rast(c(  
 "./4\_output/5. EU\_hig\_res\_CV\_PRED\_future/Run\_std.tif",  
 "./4\_output/6. WORLD\_hig\_res\_CV\_PRED\_future/Run\_std.tif"  
)) %>% mean()

#### Final stack

final\_res <- c(Fut\_EU\_mean, Fut\_WORLD\_mean, final\_map, Fut\_ALL\_std)  
  
names(final\_res) <- c("EU\_mean", "EU\_thresh", "WORLD\_mean", "WORLD\_thresh", "final", "std")  
terra::writeRaster(final\_res, "./4\_output/Result/FUTURE\_Final\_map.tif", overwrite = T)

### Germany Maps

rast("./4\_output/Result/TODAY\_Final\_map.tif") %>%  
 crop(AOI[AOI$Name == "DEU", ]) %>%  
 mask(AOI[AOI$Name == "DEU", ], filename = "./4\_output/Result/TODAY\_Final\_map\_GERMANY.tif", overwrite = TRUE)  
  
rast("./4\_output/Result/FUTURE\_Final\_map.tif") %>%  
 crop(AOI[AOI$Name == "DEU", ]) %>%  
 mask(AOI[AOI$Name == "DEU", ], filename = "./4\_output/Result/FUTURE\_Final\_map\_GERMANY.tif", overwrite = TRUE)

### Plots

AOI <- st\_read("./3\_ready/AOI\_border.shp")  
  
download.file(  
 url = "https://daten.gdz.bkg.bund.de/produkte/vg/vg2500/aktuell/vg2500\_12-31.tm32.shape.zip",  
 destfile = "./1\_input/German\_states.zip"  
)  
  
unzip(  
 "./1\_input/German\_states.zip",  
 exdir = "./1\_input/German\_States"  
)  
  
DE\_districts <- st\_read("./1\_input/German\_States/vg2500\_12-31.tm32.shape/vg2500/VG2500\_LAN.shp") %>%  
 st\_transform(st\_crs(AOI))

FIXME: DE\_Districts zum laufen bringen

# Europe  
# my\_col <- c("#F2F2F2", "#eea255", "#3daa62")  
# my\_col <- c("#a6cee3", "#1f78b4", "#b2df8a")  
# my\_col <- c("#8da0cb", "#66c2a5", "#fc8d62")  
# my\_col <- c("#ffffbf", "#fc8d59", "#91bfdb")  
# my\_col <- c("#edf8b1", "#7fcdbb", "#2c7fb8")  
my\_col <- c("#ffffff", "#99d8c9", "#1e854b")  
my\_comp\_type <- "rose"  
my\_comp\_lable <- 3  
my\_comp\_size <- 5  
my\_comp\_pos <- c("right", "top")  
  
  
final\_res\_curr <- rast("./4\_output/Result/TODAY\_Final\_map.tif")  
final\_res\_fut <- rast("./4\_output/Result/FUTURE\_Final\_map.tif")  
  
my\_bbox <- matrix(c(-1275875, 4470206, 3696405, 7907273), byrow = F, nrow = 2) %>% bbox()  
  
curr\_fin <- tm\_shape(final\_res\_curr[["final"]], bbox = my\_bbox) + tm\_raster(palette = my\_col, title = "A) Current suitability") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2)  
  
fut\_fin <- tm\_shape(final\_res\_fut[["final"]], bbox = my\_bbox) + tm\_raster(palette = my\_col, title = "B) Future suitability") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2)  
  
curr\_std <- tm\_shape(final\_res\_curr[["std"]], bbox = my\_bbox) + tm\_raster(style = "cont", palette = my\_col, title = "C) Current prediction \nstandard deviation") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2)  
  
fut\_std <- tm\_shape(final\_res\_fut[["std"]], bbox = my\_bbox) + tm\_raster(style = "cont", palette = my\_col, title = "D) Future prediction \nstandard deviation") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_compass(type = my\_comp\_type, show.labels = my\_comp\_lable, size = my\_comp\_size, position = my\_comp\_pos)  
  
tmap\_arrange(  
 curr\_fin,  
 fut\_fin,  
 curr\_std,  
 fut\_std,  
 nrow = 2  
)  
# Germany  
final\_res\_curr <- rast("./4\_output/Result/TODAY\_Final\_map\_GERMANY.tif")  
final\_res\_fut <- rast("./4\_output/Result/FUTURE\_Final\_map\_GERMANY.tif")  
  
  
curr\_fin <- tm\_shape(final\_res\_curr[["final"]]) + tm\_raster(palette = my\_col, title = "A) Current suitability") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2)  
  
fut\_fin <- tm\_shape(final\_res\_fut[["final"]]) + tm\_raster(palette = my\_col, title = "B) Future suitability") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2)  
  
curr\_std <- tm\_shape(final\_res\_curr[["std"]]) + tm\_raster(style = "cont", palette = my\_col, title = "C) Current prediction \nstandard deviation") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2)  
  
fut\_std <- tm\_shape(final\_res\_fut[["std"]]) + tm\_raster(style = "cont", palette = my\_col, title = "D) Future prediction \nstandard deviation") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2) +  
 tm\_compass(type = my\_comp\_type, show.labels = my\_comp\_lable, size = my\_comp\_size, position = my\_comp\_pos)  
  
tmap\_arrange(  
 curr\_fin,  
 fut\_fin,  
 curr\_std,  
 fut\_std,  
 nrow = 2  
)

# Predators

## Wolf

First, georeference the map from the paper: [Kramer-Schadt et al., 2020](https://doi.org/10.19217/skr556)

# Load georeferenced map  
wolf\_work <- rast("./2\_work/predators/wolf\_work.tif")  
  
# Crop  
  
wolf\_work %<>%  
 crop(AOI[AOI$Name == "DEU", ]) %>%  
 mask(AOI[AOI$Name == "DEU", ], filename = "./2\_work/predators/wolf\_work\_crop.tif", overwrite = T)  
  
  
wolf\_work2 <- ifel(wolf\_work %in% c(176, 234, 147, 148, 206, 197, 142, 149, 186, 139), 0, wolf\_work)  
  
  
wolf\_work2 <- wolf\_work2 %>% sum()  
wolf\_work2 %>% plot()  
  
  
# Calculation  
c(  
 wolf\_work2,  
 ifel(wolf\_work2 %in% seq(100, 900, 1), 10, 0)  
) %>% plot()  
  
ifel(wolf\_work2 %in% seq(100, 900, 1), 10, 0) %>% writeRaster("./2\_work/predators/wolf\_work3.tif", overwrite = T)  
  
  
wbt\_resample(  
 inputs = "./2\_work/predators/wolf\_work3.tif",  
 output = "./2\_work/predators/wolf\_work4.tif",  
 base = "./4\_output/Result/TODAY\_Final\_map.tif",  
 method = "cc"  
)  
  
rast("./2\_work/predators/wolf\_work4.tif") %>%  
 crop(AOI[AOI$Name == "DEU", ]) %>%  
 mask(AOI[AOI$Name == "DEU", ], filename = "./2\_work/predators/wolf\_work5.tif", overwrite = T)  
  
wolf\_work5 <- rast("./2\_work/predators/wolf\_work5.tif")  
final\_res\_curr <- rast("./4\_output/Result/TODAY\_Final\_map\_GERMANY.tif")[["final"]]  
  
wolf\_overlap <- ifel((final\_res\_curr + wolf\_work5) <= 10, 0, (final\_res\_curr + wolf\_work5))  
  
cls <- data.frame(id = c(0, 11, 12), cover = c("No Overlap/Not Suitable", "1/2 Overlap", "Both Overlap"))  
levels(wolf\_overlap) <- cls  
  
writeRaster(wolf\_overlap, "./4\_output/Predators/wolf\_overlap.tif", overwrite = T)  
  
plot(rast("./4\_output/Predators/wolf\_overlap.tif"))  
plot(DE\_districts$geometry, add = T)

## Fox

Download DOI from: [GBIF](https://doi.org/10.15468/dl.usq8mf).

download.file(  
 "https://api.gbif.org/v1/occurrence/download/request/0250155-230224095556074.zip",  
 "./1\_input/Fox\_occurrences.zip"  
)  
  
unzip(  
 "./1\_input/Fox\_occurrences.zip",  
 exdir = "./1\_input/Fox\_occurrences"  
)  
  
foxgbif <- read\_delim("./1\_input/Fox\_occurrences/0250155-230224095556074.csv", delim = "\t")  
  
foxgbif %<>%  
 select(year, countryCode, decimalLatitude, decimalLongitude) %>%  
 drop\_na() %>%  
 st\_as\_sf(coords = c("decimalLongitude", "decimalLatitude"), crs = 4326) %>%  
 st\_transform(crs = 4087) %>%  
 mutate(source = rep("GBIF", nrow(.))) %>%  
 .[AOI[AOI$Name == "DEU", ], ]  
  
st\_write(foxgbif, "./3\_ready/predators/Fox\_Occurrences.shp")  
  
wbt\_heat\_map(  
 input = "./3\_ready/predators/Fox\_Occurrences.shp",  
 output = "./4\_output/Predators/Fox\_density.tif",  
 bandwidth = 100000,  
 kernel = "triangular",  
 cell\_size = 927.6624,  
 base = "./4\_output/Predators/wolf\_overlap.tif"  
)  
  
fox\_density <- rast("./4\_output/Predators/Fox\_density.tif")  
fox\_density %<>% crop(AOI[AOI$Name == "DEU", ]) %>% mask(AOI[AOI$Name == "DEU", ], filename = "./4\_output/Predators/Fox\_density.tif", overwrite = T)

## Plot

wolf\_overlap <- rast("./4\_output/predator/wolf\_overlap.tif")  
  
wolf\_over <- tm\_shape(wolf\_overlap) + tm\_raster(palette = my\_col, title = "B) Wolf overlap") +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_compass(type = my\_comp\_type, show.labels = my\_comp\_lable, size = my\_comp\_size, position = my\_comp\_pos)  
  
fox\_density <- rast("./4\_output/predator/Fox\_density.tif")  
  
fox\_breaks <- c(0, 25, 50, 100, 200, 400, 800, Inf)  
  
fox\_dens <- tm\_shape(fox\_density) + tm\_raster(palette = my\_col, title = "C) Fox densityy", breaks = fox\_breaks) +  
 tm\_shape(DE\_districts$geometry[DE\_districts$GF != 8]) + tm\_borders(col = "black", lwd = 2) +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2) +  
 tm\_compass(type = my\_comp\_type, show.labels = my\_comp\_lable, size = my\_comp\_size, position = my\_comp\_pos)  
  
tmap\_arrange(  
 wolf\_over,  
 fox\_dens,  
 nrow = 1  
)

# Area calculations

## Europe

## Today  
final\_res\_curr <- rast("./4\_output/Result/TODAY\_Final\_map.tif")[["final"]]  
  
truesize <- terra::cellSize(final\_res\_curr, unit = "km")  
  
EU\_curr\_area <- terra::zonal(truesize, final\_res\_curr, fun = "sum") %>%  
 mutate(proportion = area / sum(area))  
  
  
## Future  
final\_res\_fut <- rast("./4\_output/Result/FUTURE\_Final\_map.tif")[["final"]]  
  
truesize <- terra::cellSize(final\_res\_fut, unit = "km")  
  
EU\_fut\_area <- terra::zonal(truesize, final\_res\_fut, fun = "sum") %>%  
 mutate(proportion = area / sum(area))  
  
  
## Overlap  
writeRaster(rast("./4\_output/Result/FUTURE\_Final\_map.tif")[["final"]], "./4\_output/Area/FUTURE\_Final\_map\_final.tif", overwrite = T)  
  
wbt\_resample(  
 input = "./4\_output/Area/FUTURE\_Final\_map\_final.tif",  
 output = "./4\_output/Area/FUTURE\_Final\_map\_final\_hires.tif",  
 base = "./4\_output/Result/TODAY\_Final\_map.tif"  
)  
  
final\_res\_fut <- rast("./4\_output/Area/FUTURE\_Final\_map\_final\_hires.tif")  
  
cls <- data.frame(id = c(0, 1, 2), cover = c("No Overlap/Not Suitable", "1/2 Overlap", "Both Overlap"))  
levels(final\_res\_fut) <- cls  
  
writeRaster(final\_res\_fut, "./4\_output/Area/FUTURE\_Final\_map\_final\_hires2.tif", overwrite = T)  
  
final\_res\_fut <- rast("./4\_output/Area/FUTURE\_Final\_map\_final\_hires2.tif")  
  
  
same\_class <- final\_res\_curr == final\_res\_fut  
  
truesize <- terra::cellSize(same\_class, unit = "km")  
  
same\_class\_area <- terra::zonal(truesize, same\_class, fun = "sum") %>%  
 mutate(proportion = area / sum(area))

## Germany

final\_res\_curr <- rast("./4\_output/Result/TODAY\_Final\_map\_GERMANY.tif")[["final"]]  
  
truesize <- terra::cellSize(final\_res\_curr, unit = "km")  
  
GER\_curr\_area <- terra::zonal(truesize, final\_res\_curr, fun = "sum") %>%  
 mutate(proportion = area / sum(area))  
  
  
  
# Future  
  
final\_res\_fut <- rast("./4\_output/Result/FUTURE\_Final\_map\_GERMANY.tif")[["final"]]  
  
truesize <- terra::cellSize(final\_res\_fut, unit = "km")  
  
GER\_fut\_area <- terra::zonal(truesize, final\_res\_fut, fun = "sum") %>%  
 mutate(proportion = area / sum(area))  
  
  
  
## Overlap  
writeRaster(rast("./4\_output/Result/FUTURE\_Final\_map\_GERMANY.tif")[["final"]], "./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final.tif", overwrite = T)  
  
wbt\_resample(  
 input = "./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final.tif",  
 output = "./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final\_hires.tif",  
 base = "./4\_output/Result/TODAY\_Final\_map\_GERMANY.tif"  
)  
  
final\_res\_fut <- rast("./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final\_hires.tif")  
  
cls <- data.frame(id = c(0, 1, 2), cover = c("No Overlap/Not Suitable", "1/2 Overlap", "Both Overlap"))  
levels(final\_res\_fut) <- cls  
  
writeRaster(final\_res\_fut, "./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final\_hires2.tif", overwrite = T)  
  
final\_res\_fut <- rast("./4\_output/Area/FUTURE\_Final\_map\_GERMANY\_final\_hires2.tif")  
  
same\_class <- final\_res\_curr == final\_res\_fut  
  
truesize <- terra::cellSize(same\_class, unit = "km")  
  
same\_class\_area <- terra::zonal(truesize, same\_class, fun = "sum") %>%  
 mutate(proportion = area / sum(area))

## Predators

## Overlaping area  
wolf\_overlap <- rast("./4\_output/Predators/wolf\_overlap.tif")  
  
truesize <- terra::cellSize(wolf\_overlap, unit = "km")  
  
Wolf\_area <- terra::zonal(truesize, wolf\_overlap, fun = "sum") %>%  
 mutate(proportion = area / sum(area))

# Multivariate environmental simillarity surface (MESS)

## Load the data

load("./3\_ready/model input/PA\_AOI.Rdata")  
  
Presences.cv <- all\_data$train.df %>%  
 filter(pa == 1) %>%  
 add\_row(  
 all\_data$test.df %>%  
 filter(pa == 1)  
 )  
  
  
all.points <- all\_data$train.points %>%  
 filter(pa == 1) %>%  
 add\_row(all\_data$test.points %>% filter(pa == 1)) %>%  
 add\_row(all\_data$train.points %>% filter(pa == 0)) %>%  
 add\_row(all\_data$test.points %>% filter(pa == 0))  
  
AOI <- st\_read("./3\_ready/AOI\_border.shp")

EU\_environment <- rast("./3\_ready/EU\_environment.tif")  
  
envMESS <- EU\_environment[[  
 c("slope", "bio\_12", "bio\_1", "water\_distance", "trees")  
]] %>%  
 brick()  
  
pointsMESS <- Presences.cv %>%  
 select(c("slope", "bio\_12", "bio\_1", "water\_distance", "trees")) %>%  
 as.data.frame()  
  
mess\_raster\_high <- mess(  
 x = envMESS,  
 v = pointsMESS,  
 full = FALSE  
)  
  
mess\_raster\_high %<>% rast()  
  
writeRaster(mess\_raster\_high, "./2\_work/MESS\_unscaled.tif", overwrite = T)  
  
mess\_raster\_high <- rast("./2\_work/MESS\_unscaled.tif")  
  
mess\_extr <- terra::extract(mess\_raster\_high, all.points$geometry %>% st\_coordinates()) %>%  
 mutate(pa = all.points$pa) %>%  
 as\_tibble()  
mess\_extr %<>% .[complete.cases(.), ]  
mess\_eval <- ROCR::prediction(mess\_extr$mess, mess\_extr$pa)  
mess\_thresh <- ROCR::performance(mess\_eval, measure = "acc")@x.values[[1]][which(ROCR::performance(mess\_eval, measure = "acc")@y.values[[1]] == max(ROCR::performance(mess\_eval, measure = "acc")@y.values[[1]]))] %>%  
 unlist() %>%  
 min()  
  
mess\_raster\_high\_new <- ifel(  
 mess\_raster\_high > mess\_thresh,  
 2,  
 ifel(mess\_raster\_high < 0, 0, 1)  
)  
  
  
cls <- data.frame(id = c(2, 1, 0), cover = c("Similar", "Different", "Not in Range"))  
levels(mess\_raster\_high\_new) <- cls  
  
writeRaster(mess\_raster\_high\_new, "./4\_output/Result/MESS\_final.tif")  
  
my\_bbox <- matrix(c(-1275875, 4470206, 3696405, 7907273), byrow = F, nrow = 2) %>% bbox()  
  
my\_col <- c("#ffffff", "#99d8c9", "#1e854b")  
  
tm\_shape(mess\_raster\_high\_new, bbox = my\_bbox) + tm\_raster(palette = my\_col, legend.show = T, title = "MESS analysis") +  
 tm\_shape(AOI) + tm\_borders(col = "black", lwd = 2)

# Vaiable range plots between Europe and Asia

most.important <- c("bio\_1", "bio\_12", "trees", "water\_distance", "slope")  
  
presence.points <- all.points[all.points$pa == 1, ] %>%  
 .[AOI[AOI$Name %in% c("CHN", "TWN"), ], ]  
presence.points %<>%  
 rownames() %>%  
 as.numeric()  
TWN.env <- Presences.cv[presence.points, ] %>%  
 select(all\_of(most.important))  
  
  
presence.points <- all.points[all.points$pa == 1, ] %>%  
 .[AOI[AOI$Name %!in% c("CHN", "TWN"), ], ]  
presence.points %<>%  
 rownames() %>%  
 as.numeric()  
other.env <- Presences.cv[presence.points, ] %>%  
 select(all\_of(most.important))  
  
  
  
  
all.melt <- melt(other.env) %>%  
 rbind(melt(TWN.env)) %>%  
 mutate(  
 country = c(  
 rep(  
 "Europe",  
 nrow(other.env) \* ncol(other.env)  
 ),  
 rep(  
 "Asia",  
 nrow(TWN.env) \* ncol(TWN.env)  
 )  
 )  
 )  
  
all.melt %<>%   
 as\_tibble() %>%  
 mutate(variable = case\_when(  
 variable == "slope" ~ "Slope",  
 variable == "bio\_12" ~ "Mean Precipitation",  
 variable == "bio\_1" ~ "Mean Temperature",  
 variable == "water\_distance" ~ "Water Distance",  
 variable == "trees" ~ "Tree Density"  
 )  
 )  
  
g\_plot1 <- grouped\_ggbetweenstats(  
 data = all.melt,  
 x = country,  
 y = value,  
 grouping.var = variable,  
 map\_signif\_level = T,  
 pairwise.display = "all",  
 type = "nonparametric",  
 plot.type = "violin",  
 point.args = list(alpha = 0),  
 violin.args = list(aes(col = country), fill = NA, scale = "width", adjust = 1 / 2),  
 results.subtitle = T,  
 # subtitle = list("hi"),  
 xlab = "Country",  
 ylab = "Value",  
 centrality.plotting = T,  
 centrality.label.args = list(alpha = 0),  
 centrality.point.args = list(col = "black", size = 4)  
)  
  
g\_plot2 <- list()  
for (i in 1:5) {  
 g\_plot2[[i]] <- extract\_subtitle(g\_plot1[[i]])  
}  
  
for (i in 1:5) {  
 g\_plot1[[i]] <- g\_plot1[[i]] + labs(subtitle = g\_plot2[[i]][c(1:3, 7)])  
}  
  
g\_plot1

# World/AOI Plot

world\_border <- st\_read("./3\_ready/WORLD\_border.shp")  
AOI <- st\_read("./3\_ready/AOI\_border.shp")  
  
my\_col <- c("#ffffff", "#99d8c9", "#1e854b")  
  
  
tm\_shape(world\_border) +  
 tm\_fill(my\_col[1]) +  
 tm\_borders("black") +  
 tm\_shape(AOI[AOI$Name %!in% c("CHN", "TWN"), ]) +  
 tm\_fill(my\_col[3]) +  
 tm\_shape(AOI[AOI$Name %in% c("CHN", "TWN"), ]) +  
 tm\_fill(my\_col[2]) +  
 tm\_add\_legend(  
 type = "fill",  
 col = my\_col[-1],  
 labels = c("Endemic distribution", "Area of interest, Window of observation and invasive distribution"),  
 title = "Muntjak Occurrences"  
 ) +  
 tm\_add\_legend(  
 type = "line",  
 col = c("#0400ff", "black"),  
 labels = c("Conservative absence area", "Non-Conservative absence area"),  
 lwd = 2,  
 title = "Absence Areas"  
 ) +  
 tm\_shape(AOI %>% st\_union()) +  
 tm\_borders("#0400ff", lwd = 1) +  
 tm\_layout(legend.position = c("left", "bottom"), legend.bg.color = "white")