Ecological Niche Modelling

A tutorial

Pedro Tarroso (FCUP / BIOPOLIS-CIBIO)

21-05-2024

Contents

Preamble

This guide will help you build ecological niche models using the R package biomod2. It provides a basic guide for data preparation, model building, and ensemble.

The purpose of this text is to support the practical classes associated with the Ecological Modelling course and to serve as a guide for the code in the scripts. The guide's structure follows the structure of the code provided in the practical classes, with additional information to help students follow along more easily.

The study system is composed of three related species of European vipers: *Vipera aspis*, *Vipera latastei*, and *Vipera seoanei*, which meet in the Iberian Peninsula in a narrow three-way contact zone. In this guide, we will model each of these species and predict the contact zone where the three vipers intersect. Reptiles are often considered good study systems for ecological niche modeling because they rely on external factors such as temperature to regulate their internal temperature. Most of our predictor variables will be climate-related, but we will also include a vegetation-related variable. Since climate variables are available for different time periods, we will use ecological niche models to answer the following question:

- How will climate change impact the contact zone between the three vipers in Iberia?

Although the code is divided into several scripts for logical clarity, it could also be written as a single long script. In this document, I have opted to divide the chapters according to each script. For example, script 01 will be discussed in chapter 1, and so on.

Disclaimer: Many methods and processes have been simplified for teaching purposes. While the code provides a basic and correct modeling approach, it may lack certain details and additional efforts needed to make the models more robust. These points will be noted where applicable in the text.

NOTE: This is a work in progress. If you find any errors, please comment and send them back to me!

Course folder structure and data

To follow this guide you should have a folder with data and scripts. You can find the scripts in the github repository of the tutorial. Clone the repository to your computer.

You should have 9 scripts:

- 01_Presence_Data.R
- 02_ProcessRasters.R
- 03_OptimizePresenceData.R
- 04_StaticalVariableSelection.R
- 05_PrepareProjectionVars.R
- 06_ModelBuiding.R
- 07_ensembleModels.R
- 08_projectAll.R
- 09_prepareFinalMaps.R

Each chapter of this tutorial corresponds to a script listed above. Although the sequence of the scripts is important due to dependencies on data generated in previous steps, they are designed to work independently (such as in separate R sessions). At the beginning of each script, you will load the necessary data to proceed. If you follow the full tutorial sequentially in one session, you may find that some lines of code are redundant (e.g., like reopening the same library each time or reloading previously built models to make projections when they are already available in the same R session).

You should also have two folders:

- data: where all the needed data is saved and where scripts are generating data
- models: where model data from Biomod will be saved

NOTE: there should be some folders named **original** within the data. These will have the *raw* data used for the modelling scripts that is obtained along the tutorial. Do not change these folders.

The **data** folder should have data organised in three folders

- data
  - other : Accessory data need for some scripts
    1. World countries shapefile from NaturalEarth
    2. Iberian Peninsula shapefile (processed from above file)
  - rasters: Here are the predictor variables to be used in the modelling process and where the processed rasters will be saved.
    - original: original *raw* data (obtained in chapter 2 and 5).
      - climate: will store 19 Bioclimatic variables for present and future predictions.
      - evi: Enhanced Vegetation Index obtained from the OpenGeoHub Foundation though OpenLandMap.
  - species: Species data as downloaded from GBIF and where processed presence data will be saved
    - original: Should have 3 zip files (one for each species of viper) as downloaded from GBIF and the extracted CSV text file with data and correctly named with species name (Vaspis, Vlatastei, Vseoanei).

The Bioclimatic climate variables are processed from monthly climate data (monthly minimum, mean and maximum temperatures and monthly precipitation) and coded as follows:

- BIO1: Annual Mean Temperature
- BIO2: Mean Diurnal Range (Mean of monthly (max temp - min temp))
- BIO3: Isothermality (BIO2/BIO7) ($\times 100$)
- BIO4: Temperature Seasonality (standard deviation $\times 100$)
- BIO5: Max Temperature of Warmest Month
- BIO6: Min Temperature of Coldest Month
- BIO7: Temperature Annual Range (BIO5-BIO6)
- BIO8: Mean Temperature of Wettest Quarter
- BIO9: Mean Temperature of Driest Quarter
- BIO10: Mean Temperature of Warmest Quarter
- BIO11: Mean Temperature of Coldest Quarter
- BIO12: Annual Precipitation
- BIO13: Precipitation of Wettest Month
- BIO14: Precipitation of Driest Month
- BIO15: Precipitation Seasonality (Coefficient of Variation)
- BIO16: Precipitation of Wettest Quarter
- BIO17: Precipitation of Driest Quarter
- BIO18: Precipitation of Warmest Quarter
- BIO19: Precipitation of Coldest Quarter

The EVI is an index derived from satellite imagery that reflects landscape productivity related to vegetation content by measuring the *greenness*. The original data is bi-monthly summarized (resulting in 1 variable each pair of month), and in this case, it has been further summarized into a single file representing the annual maximum value achieve for 2020.

Species data were collected from GBIF without any filtering. The doi of the data citation are:

- *V. aspis*: doi:10.15468/dl.wmv6q6
- *V. latastei*: doi:10.15468/dl.fa23t9
- *V. seoanei*: doi:10.15468/dl.cvfxu6

All scripts start by setting the **working directory**. This is important because all paths in the scripts are relative to this directory. For example, if you placed your folder "Practical_EcoMod" on the desktop, the paths would be:

- Windows: C:\Users\Peter\Desktop\Practical_EcoMod
- Mac /Users/Peter/Desktop/Practical_EcoMod
- Linux: /home/peter/Desktop/Practical_EcoMod

therefore the full path to the file `data/other/iberia.shp` would be:

- Windows: C:\Users\Peter\Desktop\Practical_EcoMod\data\other\iberia.shp
- Mac /Users/Peter/Desktop/Practical_EcoMod/data/other/iberia.shp
- Linux: /home/peter/Desktop/Practical_EcoMod/data/other/iberia.shp

by setting the working directory as:

- Windows (notice the use of double \\ ):

```
setwd("C:\\Users\\Peter\\Desktop\\Practical_EcoMod")
```

- Mac :

```
setwd("/Users/Peter/Desktop/Practical_EcoMod")
```

- Linux :

```
setwd("/home/peter/Desktop/Practical_EcoMod")
```

Once the working directory is set, all files can be referenced relative to this directory. For example, instead of providing the full path to the file, you can simply use the relative path`data/other/iberia.shp` for R to locate the file.

Each individual script begins by setting this working directory, but in this document, this step is omitted after being set once here.

Chapter 1 - The presence data housekeeping

In this chapter, we will open and clean the presence data for each of the three species. Cleaning the data involves removing erroneous entries such as those with missing coordinates, incorrect coordinates (e.g., outside the species' known range), imprecise locations (high uncertainty), and other issues. The quality filters applied to the presence data will be based on both geographical information and attribute information provided by GBIF for each presence point.

We need a special package called CoordinateCleaner that provides functions for cleaning presence data. If you don't have it installed, you can install it either through the RStudio menu or by running the command:

```
install.packages("CoordinateCleaner")
```

NOTE: Besides here, only in Chapter 6 will you be asked to install some packages in this document. However, be aware that if a package is not installed, R will generate an error indicating that the package cannot be found or is not available. In such cases, install the necessary package using the instruction provided above (changing the name of the package to the one you need!).

Packages only need to be installed once. However, to use the functionality of an installed package in an R session, you must load it with the `library()` command. This command needs to be executed in every new R session where you want to use the additional package.

open the package

```
library(CoordinateCleaner)
```

We need to open the raw GBIF downloaded data into R. Since we downloaded it in the simple text CSV format, we can use native R functions to open it. If you check the data with a simple text editor, you would see that it has a header (the first line has column names), and each column is separated by a TAB. We specify these settings in the arguments of the `read.table()` function to correctly read the data. Additionally, we set two other arguments: `quote=""` and `comment.char=""`. These settings prevent the possible use of characters such as double quotes (“) or hash symbols (#) in the data, which could potentially break the reading process.”.

```
vasp <- read.table("data/species/original/Vaspis.csv", sep="\t", header=TRUE, quote="")
vlat <- read.table("data/species/original/Vlatastei.csv", sep="\t", header=TRUE, quote="")
vseo <- read.table("data/species/original/Vseoanei.csv", sep="\t", header=TRUE, quote="")
```

We can keep track of the dimensions of each dataset to check how many points we remove in the process.[1]

```
dim(vasp)
```

```
## [1] 35929    50
```

```
dim(vlat)
```

```
## [1] 4477   50
```

```
dim(vseo)
```

```
## [1] 2070   50
```

Remove rows with missing coordinates

The column names that GBIF uses for coordinates are named *decimalLongitude* and *decimalLatitude*. We need to check these columns, as some of the presence data may be provided without locations.

To retrieve the rows with missing coordinate data (NA), we can create a mask that detects NAs. The function *is.na* returns TRUE if the value in the row is NA, and FALSE if it contains data. Mathematical operations involving numbers and NAs typically result in NA. We leverage this functionality to check whether any of the longitude or latitude (or both) columns contain NAs.

```
mask_vasp <- is.na(vasp$decimalLatitude + vasp$decimalLongitude)
vasp <- vasp[!mask_vasp,]
```

The exclamtion point (!) inverts the logical values: TRUE become FALSE, and FALSE become TRUE. This is useful because TRUEs are the rows preserved and we want to preserved the non-NA in the filtering with [ ].

We repeat for the other species:

[1] In RStudio, you can monitor the Environment panel, usually located at the top right corner of the interface, for updates on the dataset dimensions.

```
mask_vlat <- is.na(vlat$decimalLatitude + vlat$decimalLongitude)
vlat <- vlat[!mask_vlat,]

mask_vseo <- is.na(vseo$decimalLatitude + vseo$decimalLongitude)
vseo <- vseo[!mask_vseo,]
```

Just the dimensions again to track how many points we still have after this first filter:

```
dim(vasp)
```

```
## [1] 33813    50
```

```
dim(vlat)
```

```
## [1] 4152   50
```

```
dim(vseo)
```

```
## [1] 1862   50
```

```
# Course folder structure and da
```

Remove rows that fail cleaning tests

CoordinateCleaner package offer some data filtering. It provide a series of tests that we might select to flag with rows pass and do not pass the test. The tests we are going to use are: * capitals: if coordinates are in the country capital coordinate, it usually indicates lack of precision * centroids: if the coordinate is exactly at the centroid of a country, than it also indicate lack of precision * equal: if longitude and latitude are equal is often an error * gbif: this tests if the coordinate refers to the GBIF headquarters. * instituions: some coordinates might be attributes to the institution providing rather than to the observation location. We remove those. * zeros: if the coordinates are set to zero, (intersection of equation with prime meridian) are often wrong.

We set the test in a separate object an then run the cleaning function that will flag each row.

```
tests <- c("capitals", "centroids", "equal", "gbif", "institutions", "zeros")

flags_vasp <- clean_coordinates(x = vasp, lon = "decimalLongitude", lat = "decimalLatitude",
                                species = "species", countries = "countryCode",
                                country_refcol = "iso_a2", tests = tests)
```

```
## Testing coordinate validity
```

```
## Flagged 0 records.
```

```
## Testing equal lat/lon
```

```
## Flagged 1 records.
```

```
## Testing zero coordinates
```

```
## Flagged 1 records.
```

```
## Testing country capitals
```

```
## Flagged 65 records.
```

```
## Testing country centroids
```

```
## Flagged 121 records.
```

```
## Testing GBIF headquarters, flagging records around Copenhagen
```

```
## Flagged 0 records.
## Testing biodiversity institutions
## Flagged 11 records.
## Flagged 192 of 33813 records, EQ = 0.01.
```

If you check the last 8 columns of object "flags_vasp" you see a TRUE/FALSE columns for each test plus a *summary* that has a TRUE if the presence passed all tests. We want to keep those recods only.

```
vasp <- vasp[flags_vasp$.summary,]
```

Repeat for the other two species:

```
flags_vlat <- clean_coordinates(x = vlat, lon = "decimalLongitude", lat = "decimalLatitude",
                                species = "species", countries = "countryCode",
                                country_refcol = "iso_a2", tests = tests)
```

```
## Testing coordinate validity
## Flagged 0 records.
## Testing equal lat/lon
## Flagged 0 records.
## Testing zero coordinates
## Flagged 5 records.
## Testing country capitals
## Flagged 4 records.
## Testing country centroids
## Flagged 0 records.
## Testing GBIF headquarters, flagging records around Copenhagen
## Flagged 0 records.
## Testing biodiversity institutions
## Flagged 0 records.
## Flagged 9 of 4152 records, EQ = 0.
```

```
vlat <- vlat[flags_vlat$.summary,]

flags_vseo <- clean_coordinates(x = vseo, lon = "decimalLongitude", lat = "decimalLatitude",
                                species = "species", countries = "countryCode",
                                country_refcol = "iso_a2", tests = tests)
```

```
## Testing coordinate validity
## Flagged 0 records.
## Testing equal lat/lon
## Flagged 0 records.
## Testing zero coordinates
## Flagged 0 records.
## Testing country capitals
```

```
## Flagged 2 records.

## Testing country centroids

## Flagged 22 records.

## Testing GBIF headquarters, flagging records around Copenhagen

## Flagged 0 records.

## Testing biodiversity institutions

## Flagged 0 records.

## Flagged 24 of 1862 records, EQ = 0.01.
```

```
vseo <- vseo[flags_vseo$.summary,]
```

Remove rows based on additional GBIF information

GBIF provides uncertainty related to the precision of the coordinate. We can use this information to further filter our dataset by removing coordinates with very high uncertainty. Out models will be made with a spatial resolution of 10km (will see in later chapters) so if the coordinate is reported to have an higher uncertainty we remove it. However, not all rows have uncertainty reported. We have to preserved those rows without information as well because we don't know the precision.

```
# Remove high coordinate uncertainty
u_vasp <- vasp$coordinateUncertaintyInMeters
vasp <- vasp[u_vasp <= 10000 | is.na(u_vasp),]

u_vlat <- vlat$coordinateUncertaintyInMeters
vlat <- vlat[u_vlat <= 10000 | is.na(u_vlat),]

u_vseo <- vseo$coordinateUncertaintyInMeters
vseo <- vseo[u_vseo <= 10000 | is.na(u_vseo),]
```

We can also remove Fossil records if they are reported by GBIF:

```
# Remove fossil records
vasp <- vasp[vasp$basisOfRecord != "FOSSIL_SPECIMEN",]
vlat <- vlat[vlat$basisOfRecord != "FOSSIL_SPECIMEN",]
vseo <- vseo[vseo$basisOfRecord != "FOSSIL_SPECIMEN",]
```

For some records, an individual count is reported. This can be higher than one (for instace, how many times a species is seen in a camera trap) but it can also be zero, indicating absence. We only keep those that have 1 or more count or if this info is not reported.

```
# Sometimes there are records of absences. Remove them.
i_vasp <- vasp$individualCount
vasp <- vasp[i_vasp > 0 | is.na(i_vasp) , ]

i_vlat <- vlat$individualCount
vlat <- vlat[i_vlat > 0 | is.na(i_vlat) , ]

i_vseo <- vseo$individualCount
vseo <- vseo[i_vseo > 0 | is.na(i_vseo) , ]
```

To preserve some quality we remove very old records that might not exist anymore or might have very imprecise location. Here we arbitrarily chose 1970 to define a date of reliable records.

```
# Remove very old records
vasp <- vasp[vasp$year > 1970 | is.na(vasp$year), ]
vlat <- vlat[vlat$year > 1970 | is.na(vlat$year), ]
vseo <- vseo[vseo$year > 1970 | is.na(vseo$year), ]
```

Remove duplicated coordinates

Some of the records might be observed at the same location (e.g. same camera trapping site at different field work seasons). We want only one record per location, so we can filter out repeated location, keeping just one (preserving the uniques).

```
# Select only coordinates and remove duplicated coordinates
vasp <- data.frame(species="Vaspis", x=vasp$decimalLongitude, y=vasp$decimalLatitude)
vasp <- unique(vasp)

vlat <- data.frame(species="Vlatastei", x=vlat$decimalLongitude, y=vlat$decimalLatitude)
vlat <- unique(vlat)

vseo <- data.frame(species="Vseoanei", x=vseo$decimalLongitude, y=vseo$decimalLatitude)
vseo <- unique(vseo)
```

We can check the size again:

```
dim(vasp)
```

```
## [1] 14323     3
```

```
dim(vlat)
```

```
## [1] 2193    3
```

```
dim(vseo)
```

```
## [1] 1356    3
```

Check map

We can merge the three species in the same dataframe to have only one file.

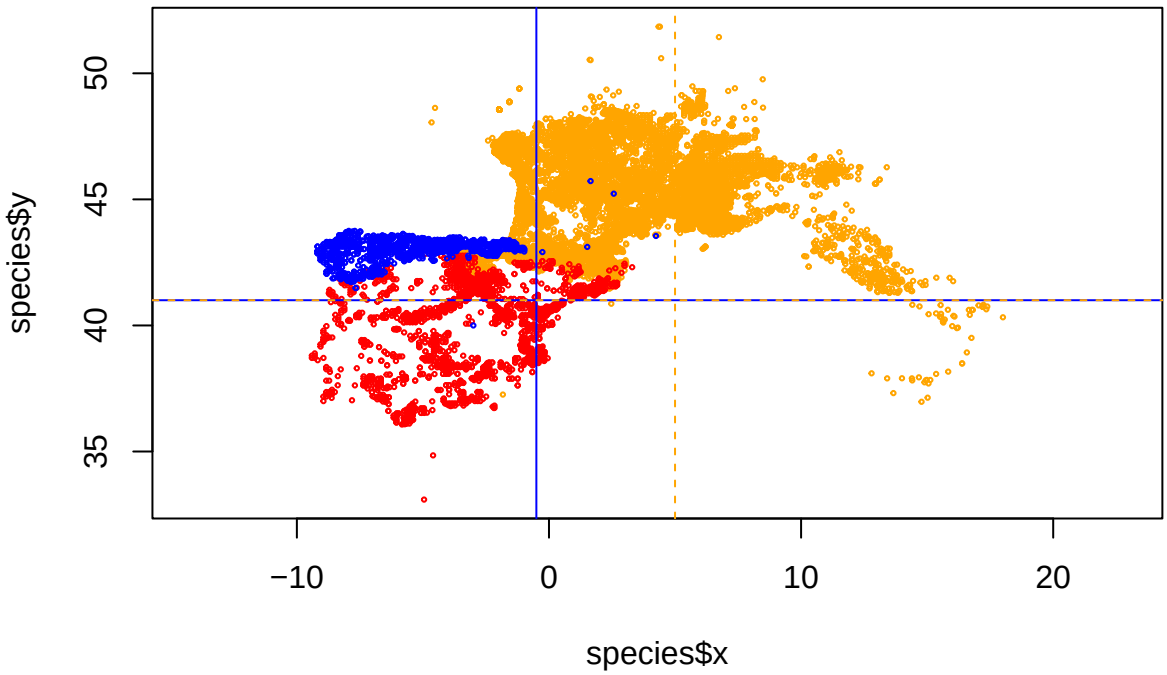
```
species <- rbind(vasp, vlat, vseo)
```

and we can plot our points to check in the map if we spot some obvious errors:

```
sp_factor <- as.factor(species$species)
levels(sp_factor)
```

```
## [1] "Vaspis"    "Vlatastei" "Vseoanei"
```

```
levels(sp_factor) <- c("orange", "red", "blue")
color <- as.character(sp_factor)
plot(species$x, species$y, asp=1, cex=0.3, col=color)

abline(v=-0.5, col='blue')
abline(h=41, col='blue')

abline(v=5, col='orange', lty=2)
abline(h=41, col='orange', lty=2)
```



As shown by the plot with the lines, *V. seoanei* in blue have a few suspicious points at the east of main distribution and also at the south, while *V. aspis* has some wrong points at south west of the main distribution. We can remove those points based on the coordinates:

For *V. soanei* everything that is at the right of -0.5º longitude OR south of 41º latitude is to remove:

```
mask <- species$species == "Vseoanei" & ( species$x > -0.5 | species$y < 41)
species <- species[!mask,]
```

For *V aspis* we remove everythin that is at left of 5ª longitude AND south of 41º latitude.

```
mask <- species$species == "Vaspis" & ( species$x < 5 & species$y < 41)
species <- species[!mask,]
```

Save the new data

We can check the final dimensions:

```
dim(vasp)
```

```
## [1] 14323     3
```

```
dim(vlat)
```

```
## [1] 2193    3
```

```
dim(vseo)
```

```
## [1] 1356    3
```

We set a file name and we write our data set:

```
filename <- "data/species/speciesPresence_v1.csv"
write.table(species, filename, sep="\t", row.names=FALSE, col.names=TRUE)
```

We saved as version 1 because we will have to further remove more data to match our model spatial resolution (Chapter 3).

Chapter 2 - Processing the raster data

In this chapter, we are going to obtain and process the predictors in the *data/raster* folder. In the process of building an ecological niche model, we need to select and retrieve variables related to the niche of our species of interest and our specific hypothesis or question. Often, these variables can be found in several public databases (e.g., WorldClim climate, satellite products) or other sources, but they usually require some level of processing. Our goal here is to obtain these variable and perform some GIS raster processing inside R to ensure uniform geographical properties among the raster layers. These steps include:

- Setting the same extent for all layers (aligning the rasters)
- Setting the same resolution (in this case, the study will be performed at a 10km resolution)
- Applying the same No Data mask to all layers

Defining the resolution and extent of the study area requires careful analysis. In this example, we set an extent that covers the combined distribution of the three species. We opt for a 10km resolution because many presence data from national atlases and other sources are available at this resolution. This allows us to use these data in our model (in the previous chapter, we removed points that had an uncertainty higher than 10km). We also choose this resolution for practical reasons. A very high resolution would require much more processing time and storage space, which may not be feasible within the available time to run this example. However, the resolution should always be carefully balanced between data availability (both predictors and presence data), processing limitations, and, most importantly, the requirements of the study system. For example, it might not make sense to study elephant distributions at a very high spatial resolution.

Note that in this chapter we are limiting the training area, i.e., the area where the models will retrieve presence and absence data to calibrate and find the optimal statistical solution. Later, in chapter 5, we will process the variables to focus only on the area of model projections, which is the Iberian Peninsula.

For this chapter, we need to obtain the 19 bioclimatic variables and the EVI variable. All files should be downloaded to *data/rasters/original* under respestive folders *climate and* evi*.

The `terra` library is essential for niche modeling. It provides functionality to handle spatial data such as rasters and vectors, as well as perform geoprocessing. This transforms R into a very competent GIS tool. The library *geodata* provides access to climate data from WorldClim and other spatial data.

```
library(terra)
library(geodata)
```

General layers

Throughout the tutorial, we will define a training area and a projection area for the models. We will explore each in detail later, but here's a brief overview: the training area is where we gather data to build the models, usually covering the species' distribution range. The projection area is where we aim to generate model predictions. In this case, the training area includes most of Western and Central Europe and part of North Africa, aligning with our focal species' distribution. The projection area is the Iberian Peninsula (Portugal and Spain), where we will analyze potential contact zones.

Obtain a vector of country polygons with geodata and save it in the appropriate folder.

```
countries <- world(path = "data/other")
```

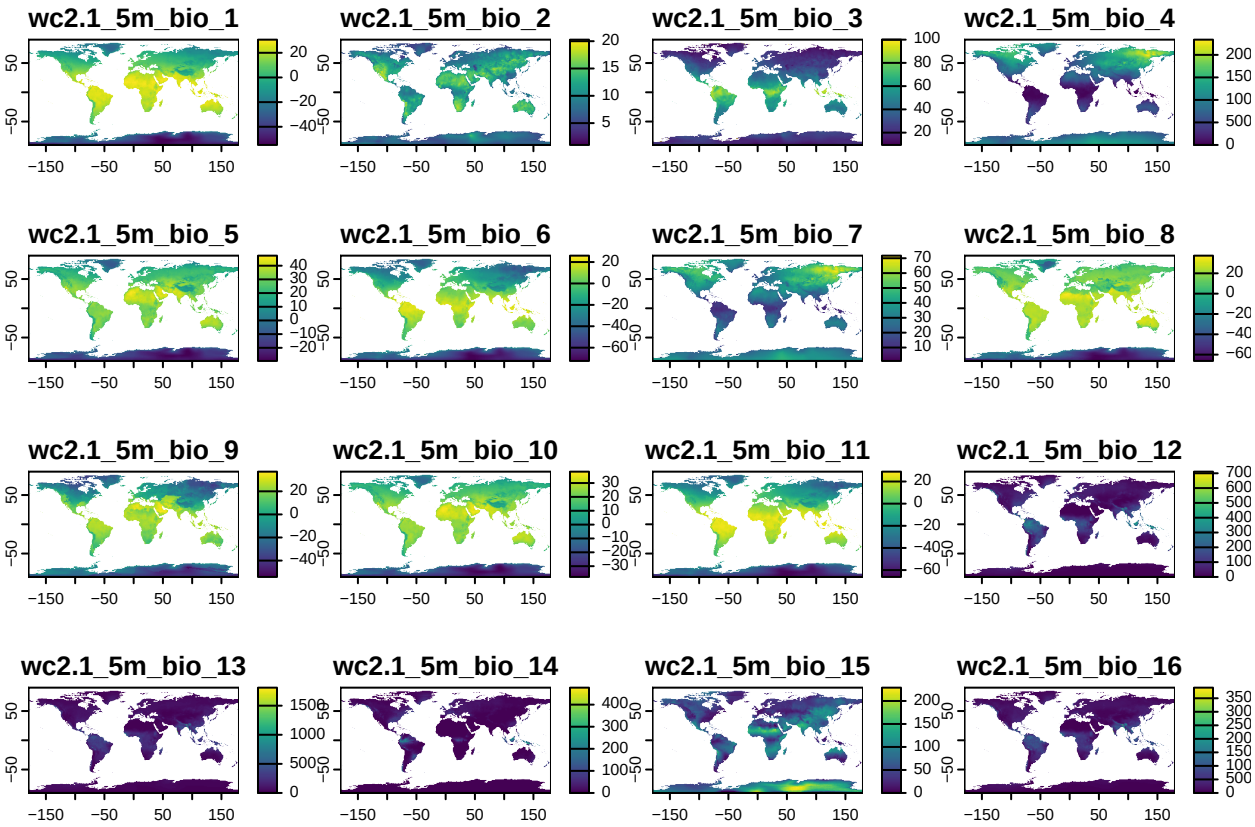
Now we use the information of the countries within the layer to get our projection area. Since with want only the continental landmass, we have to separate (disaggregate) the polygons with `disagg` command and keep the two largest polygons.

```
iberia <- countries[countries$GID_0 %in% c("PRT", "ESP")]
iberia <- disagg(iberia)
areas <- expanse(iberia)
iberia <- iberia[order(areas, decreasing=TRUE)[1:2]]
writeVector(iberia, "data/other/iberia.shp", overwrite=TRUE)
```

Processing climate layers

First, we are going to obtain the layers from Worldclim through geodata library. These are the 19 bioclimatic variables (see above for details). The function allow us to set a priori the resolution we want (5 arc minutes = 0.083333(3) degrees ~ 10km) and the path where to write them.

```
clim <- worldclim_global(var = 'bio', res = 5, path = 'data/rasters/original')
plot(clim)
```

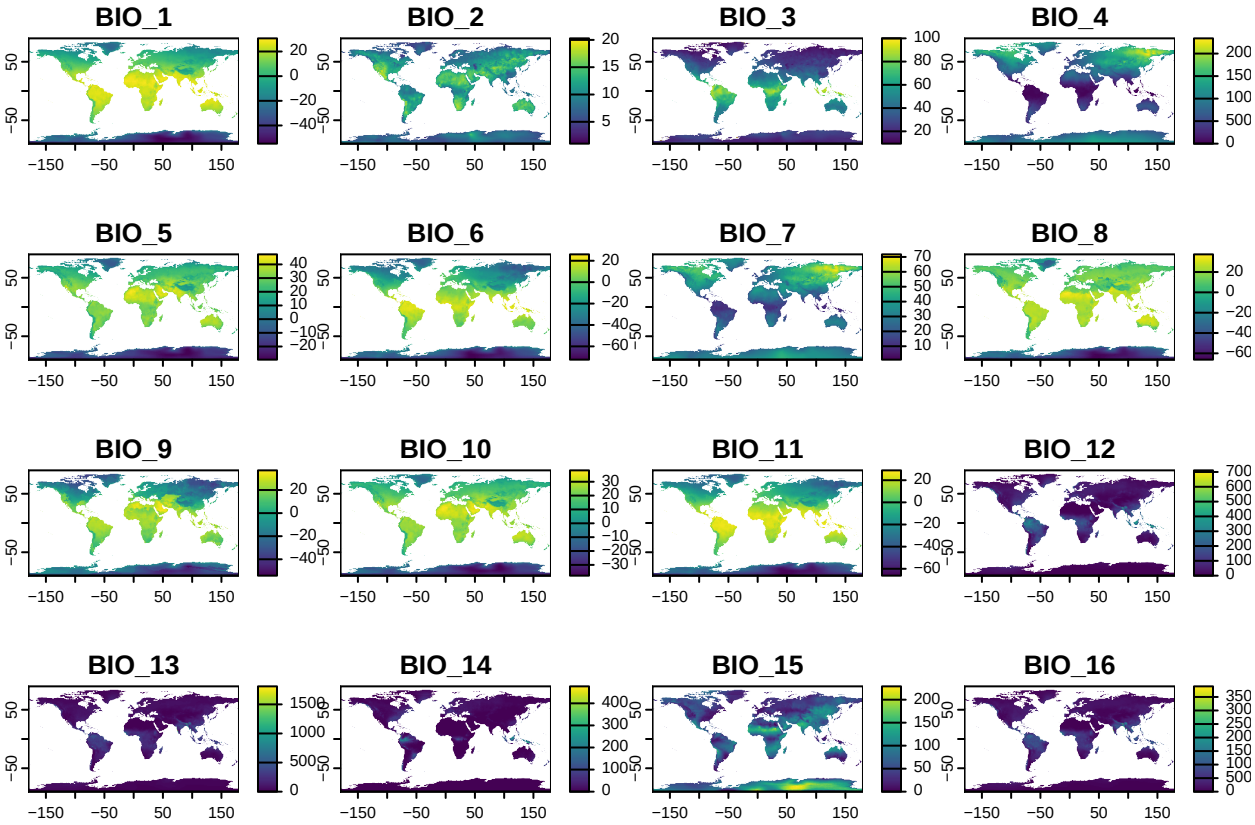


The original layer names are lengthy and complex. Since we'll frequently reference these variables by name throughout the tutorial, we'll simplify them with easily identifiable names. Given that we're opening all files sequentially from bio1 to bio19, we can rename the layers accordingly.

```
names(clim)
```

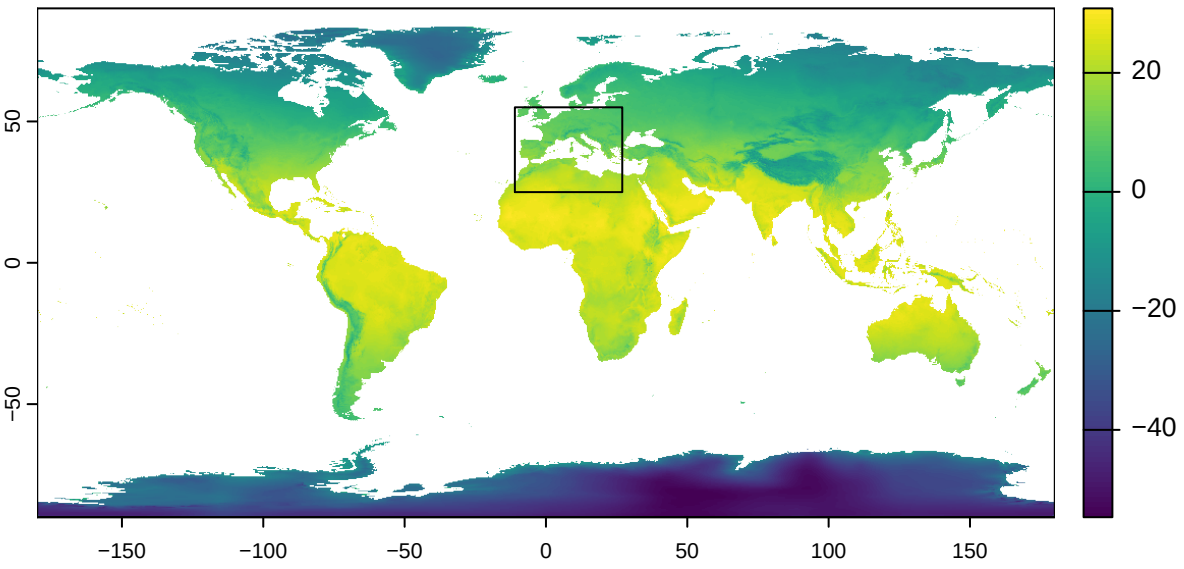
```
##  [1] "wc2.1_5m_bio_1"  "wc2.1_5m_bio_2"  "wc2.1_5m_bio_3"  "wc2.1_5m_bio_4"
##  [5] "wc2.1_5m_bio_5"  "wc2.1_5m_bio_6"  "wc2.1_5m_bio_7"  "wc2.1_5m_bio_8"
##  [9] "wc2.1_5m_bio_9"  "wc2.1_5m_bio_10" "wc2.1_5m_bio_11" "wc2.1_5m_bio_12"
## [13] "wc2.1_5m_bio_13" "wc2.1_5m_bio_14" "wc2.1_5m_bio_15" "wc2.1_5m_bio_16"
## [17] "wc2.1_5m_bio_17" "wc2.1_5m_bio_18" "wc2.1_5m_bio_19"
```

```
names(clim) <- paste0("BIO_", 1:19)
plot(clim)
```



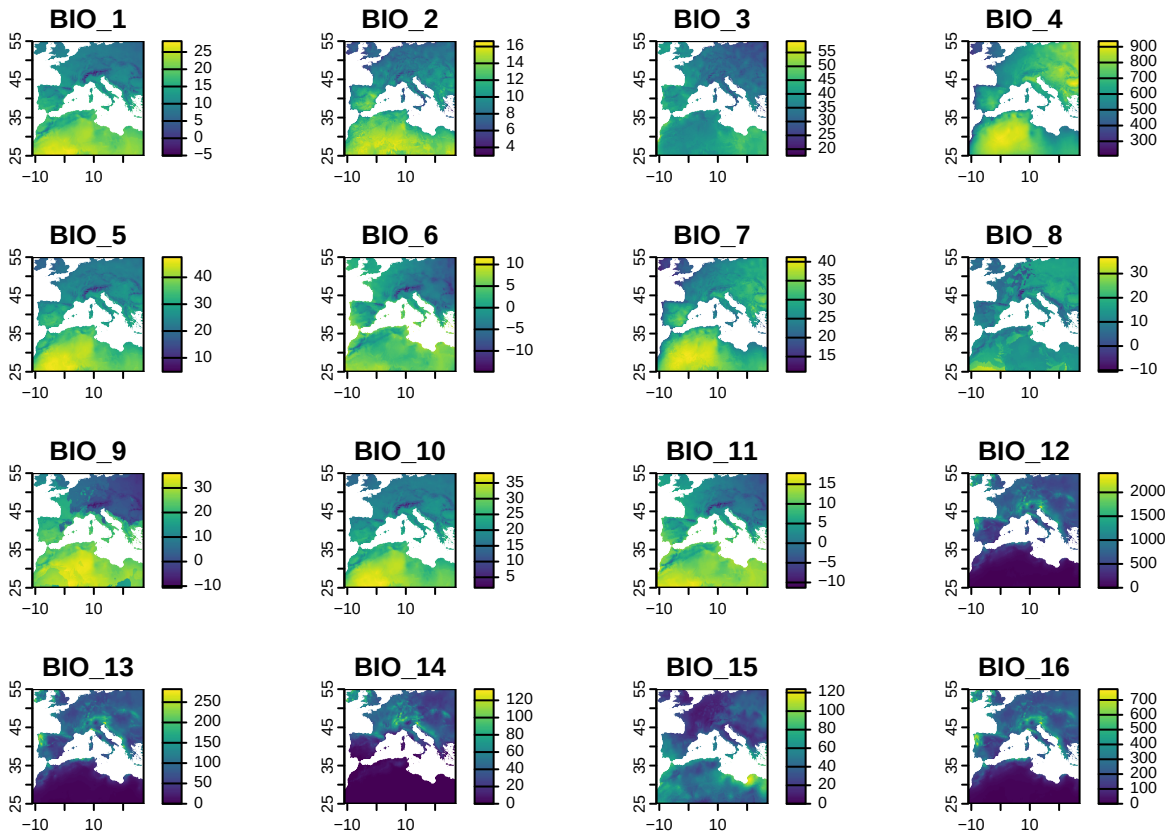
The study area (training area for models) needs to encompass the full distribution of the three species. We could either check the extreme coordinates of the presence data or use coarse distribution polygons (e.g., IUCN distributions) to define the study area. Here, we opt for a simpler approach by defining a rectangle that we know includes the species' distributions. This rectangle extends from -11.0 to 27.0 degrees longitude and from 25 to 55 degrees latitude.

```
e <- ext(-11, 27, 25, 55)
plot(clim[[1]])
plot(e, add=T)
```



We can now use the rectangle extent to crop the climate layers.

```
clim <- crop(clim, e)
plot(clim)
```



Process EVI

The Enhanced Vegetation Index (EVI) indicates land productivity and serves here as a continuous variable, acting as a proxy for habitat. We will use 2020 data available from the OpenGeoHub website, which hosts a variety of free spatial data useful for ecological niche modeling. The EVI data is global with a 250m resolution, resulting in large file sizes. We need to manually download the six files for 2020 and save them in the data/rasters/evi folder. Once processed, these files can be deleted to save space. The links to each file are provided in the table.

Name	Month
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20200101_20200228_go_epsg.4326_v20230608	Jan-Feb
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20200301_20200430_go_epsg.4326_v20230608	Mar-Apr
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20200501_20200630_go_epsg.4326_v20230608	May-Jun
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20200701_20200831_go_epsg.4326_v20230608	Jul-Aug
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20200901_20201031_go_epsg.4326_v20230608	Sep-Oct
evi_mod13q1.tmwm.inpaint_p.90_250m_s_20201101_20201231_go_epsg.4326_v20230608	Nov-Dec

After downloading to the correct folder, we can open the rasters in R.

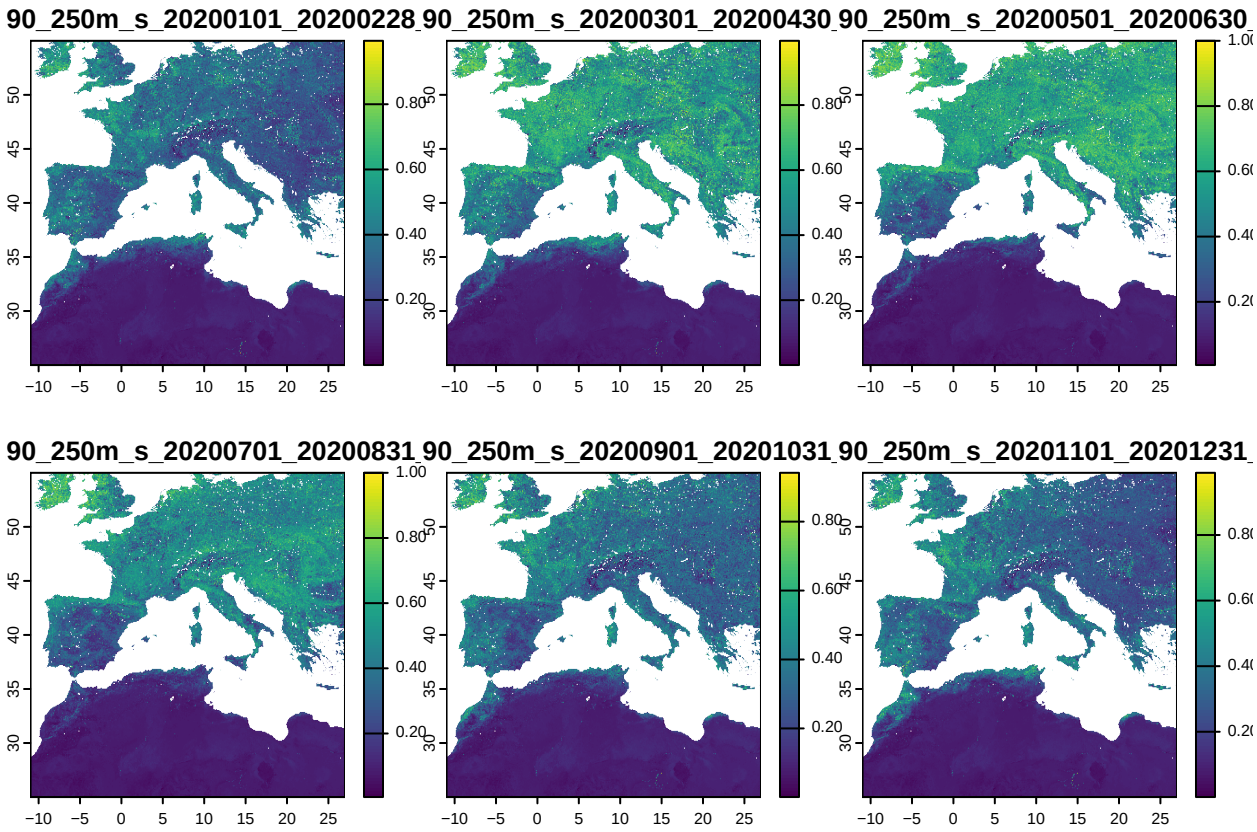
```
evifiles <- list.files("data/rasters/original/evi/", "*tif", full.names=TRUE)
evi <- rast(evifiles)
```

As they are huge files, to avoid processing unneeded areas, we crop to the study area defined above.

```
evi <- crop(evi, e)
```

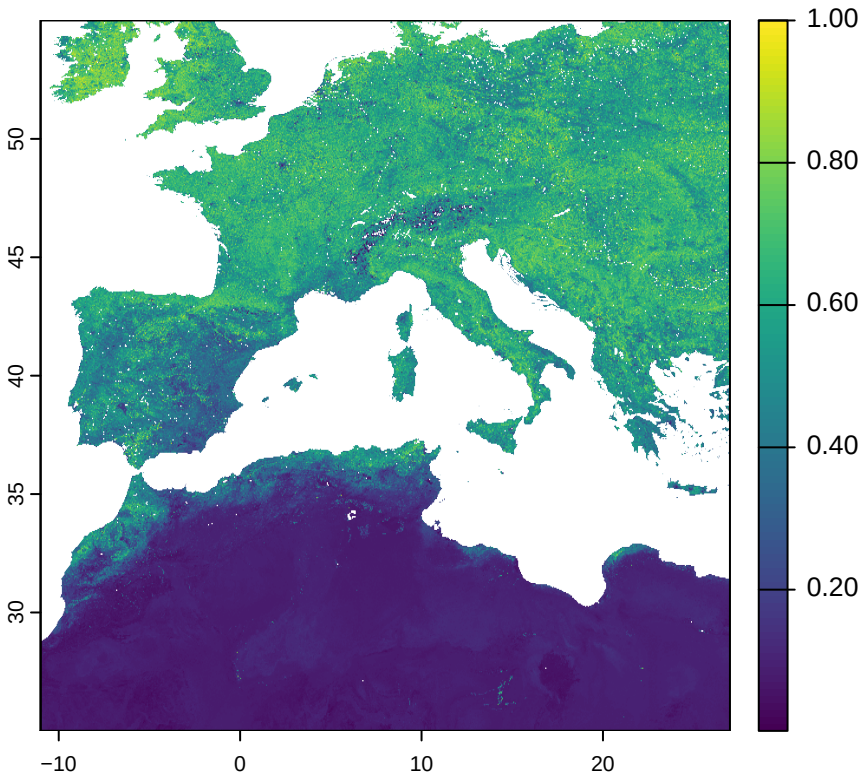
```
## |---------|---------|---------|---------|=========================================
```

```
plot(evi)
```



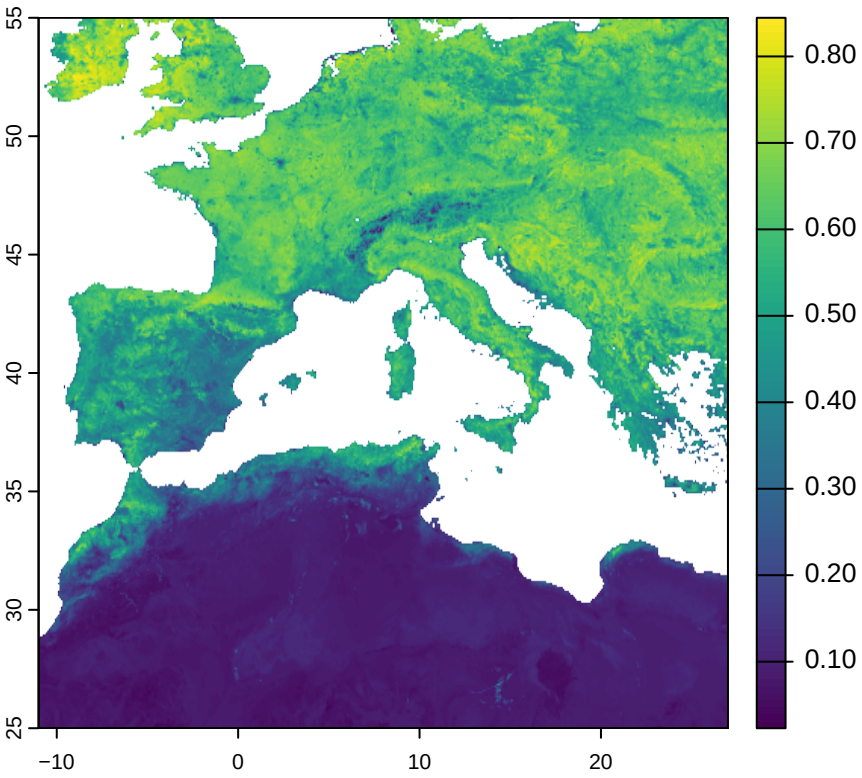
We need to further summarize the data into a single file by obtaining the maximum EVI value per pixel for the year 2020. We can use the `app` function from terra, which applies a function to summarize all layers on a pixel-by-pixel basis.

```
evi <- app(evi, max, na.rm=T)
plot(evi)
```



We still need to change the resolution of the EVI to match the climate data. However, the original resolution of the EVI (0.002245788 degrees) does not allow aggregation by an integer factor. Therefore, we need to use another function to resample the data to a new resolution. By providing the climate raster to this function, it resamples the EVI to match the exact spatial properties of the climate dataset. We use the "average" method to the resampling process, so each group of pixels is summarised by average to the new, larger one.

```
evi <- resample(evi, clim, "average")
plot(evi)
```



We can also update the layer name:

```
names(evi)
```

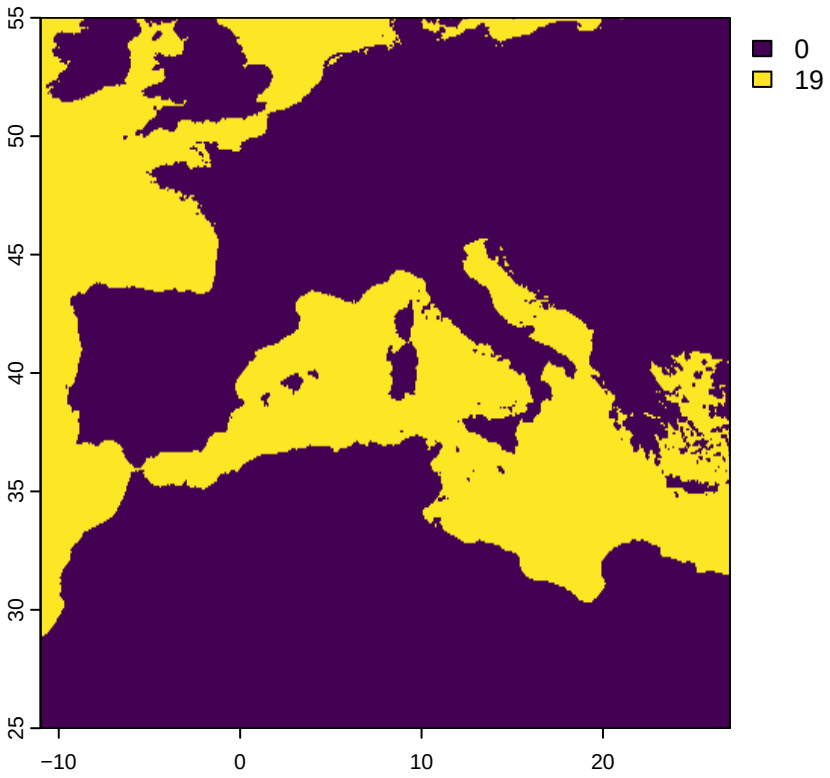
```
## [1] "max"
```

```
names(evi) <- "evi"
```

Although it is not mandatory, it is good practice to have the same No Data mask for all variables used in modeling. We need to check the No Data areas of each layer and set them to a common area. The climate layers already share the same No Data mask since they were created using the same process. However, it is not guaranteed that the EVI has the same mask. Here, any pixel set as No Data in a single layer will be set as No Data in all layers.

Let's extract and check the combination of all No Data masks. The `app` command applies a function (in this case, the `sum` function) to all corresponding pixels in all layers. By using `is.na()`, we get TRUE or FALSE if the pixel is NA or not in each layer. By summing those values, we will know in how many layers that pixel is set to NA.

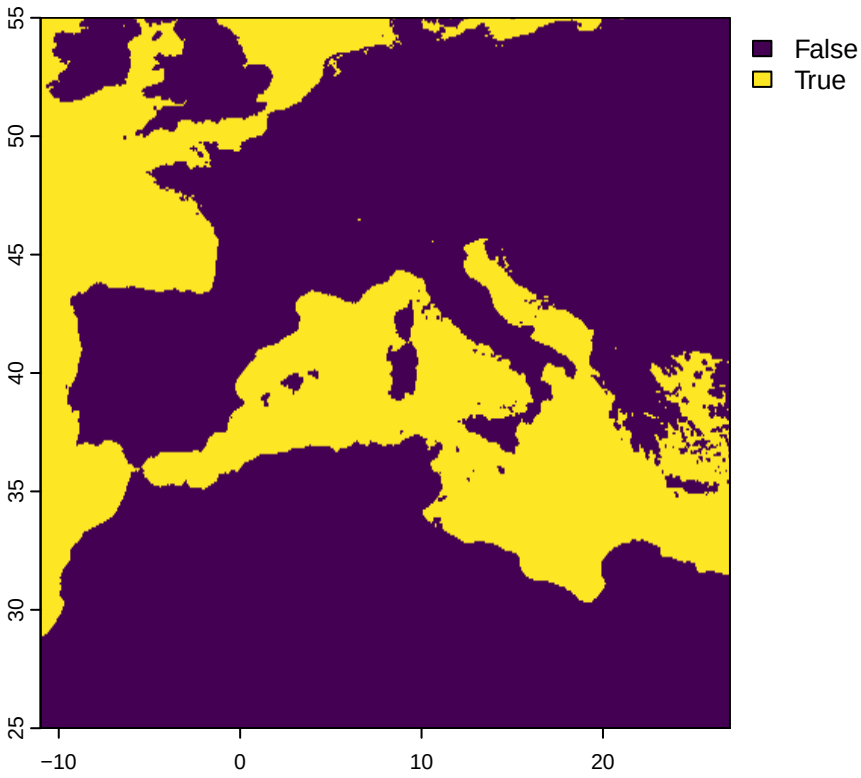
```
na.clim <- app(is.na(clim), sum)
plot(na.clim)
```



As expected, the pixel is either set to NA in all 19 layers or it has data in all.

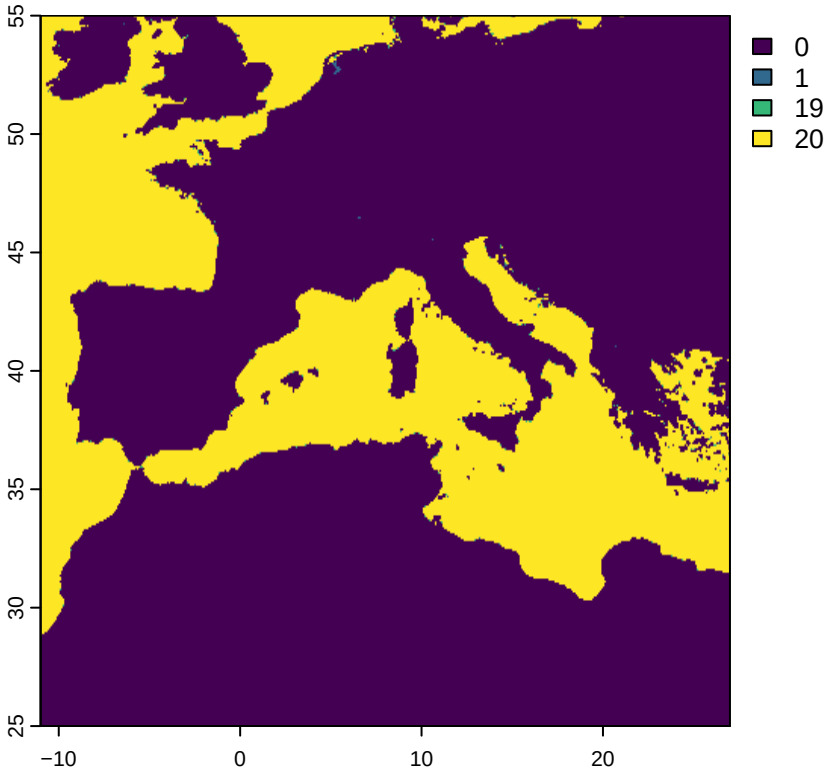
Now we sum the EVI No Data to check if there are other pixels with No Data. If so, we will have to set a common mask for all layers.

```
na.evi <- is.na(evi)
plot(na.evi)
```



Plot the sum of both masks:

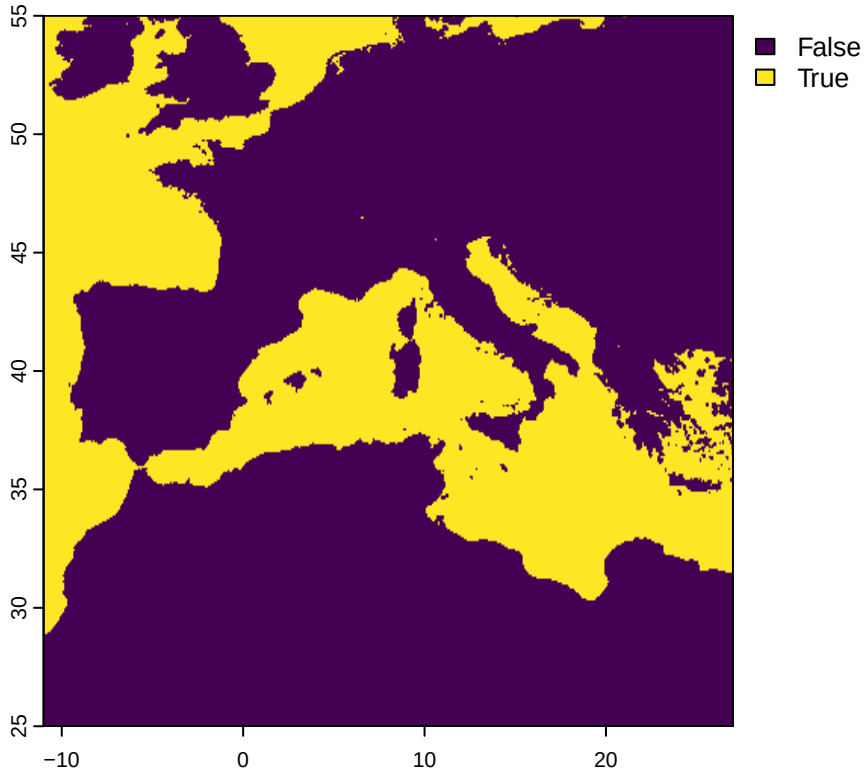
```
plot(na.clim + na.evi)
```



Notice that a few pixels have the value of 1. This means that only the EVI had those pixels set as No Data.

We need to set a final mask where we detect pixels set to NA in, at least, one layer and apply the mask to all rasters processed.

```
na.mask <- (na.clim + na.evi) > 0
plot(na.mask)
```




```
clim[na.mask] <- NA
evi[na.mask] <- NA
```

We have finished the processing of the raster variables. The climate and EVI data sets are now fully aligned and with the same resolution. We just need to save them into different TIF files[2].

```
# Write Rasters to file
# Chelsa is a single file with 19 bands/layers
writeRaster(clim, "data/rasters/climate.tif", overwrite=TRUE)
# EVI is a single file with single band/layer
writeRaster(evi, "data/rasters/evi.tif", overwrite=TRUE)
```

Note that these files a full raster georeferenced images. The images can be visualised as well in any GIS (e.g. QGIS).

Chapter 3 - Final optimization of presence data

In this chapter, we will load the presence data we worked on in Chapter 1 and the raster variables we processed in Chapter 2. We will optimize the presence data by considering the spatial resolution of the variables.

This ensures that no presence is repeated in the same pixel (also referred to as a cell). Having multiple presences in the same pixel would mean that we would provide the same set of environmental conditions repeated as many times as the presences. This can create an artificial bias in the models towards these

[2]We use the "overwrite=TRUE" just in the case some version of the files were already present and to avoid the error

values. By setting a resolution and removing duplicate presences, we also reduce bias towards areas with more sampling effort (an area with more effort tend to have more duplicates at wider spatial resolutions).

We will only need `terra` library for this chapter.

```
library(terra)
```

We can now open the relevant data for this chapter. Since in the last chapter we fully aligned the raster variables, we only need to open a single one now as it will provide enough information for this process. The presence data set is that produced at the end of chapter 1.

```
evi <- rast("data/rasters/evi.tif")
pres <- read.table("data/species/speciesPresence_v1.csv", sep="\t", header=TRUE)
```

Now we will extract raster values at each presence location point. For that, we will use the `extract` function provided by the `terra` package. This function will detect in which pixel/cell the presence is located and extract that information. If a presence is located in a No Data pixel, we will be able to identify it and remove it. However, here we are also interested in identifying if the pixel is the same or not. We cannot rely on EVI values for that, as different pixels might have the same value.

We can set the `cells=TRUE` parameter in the `extract` function. For each presence, this will provide a unique pixel/cell identifier. Thus, if two presences are in the same pixel, they will have the same identifier, and we can keep only one, removing the duplicates.

```
dt <- extract(evi, pres[,c("x", "y")], cells=TRUE)
head(dt)
```

```
##   ID       evi  cell
## 1  1 0.5606292 70845
## 2  2 0.5706229 69470
## 3  3 0.5267091 71757
## 4  4 0.6034736 68556
## 5  5 0.5350373 70382
## 6  6 0.4996728 68569
```

Now we remove those presences that fall in No Data by checking the *evi* column of the extracted data table:

```
mask <- is.na(dt$evi)

# how many presences fall in missing data?
sum(mask)
```

```
## [1] 10
```

```
dt <- dt[!mask,]
pres <- pres[!mask,]
```

Now that both the *presence* and *extrated* datasets are free of missing data, we need to check for pixel duplicates. However, there is a detail that adds a bit of complexity to the process. Since we are working with three species, we have to check for duplicates independently for each species. This is because the three species can coexist in the same pixel (sympatry), and we do not want to confound this situation as pixel duplicates.

The easiest way to do this is to create a loop over the species so that we can detect duplicates for each species independently. The code is organized as follows: 1. Check the names of the species to loop over (three different species in this case) 2. Create a column in the *presence* data set that will store a value of TRUE is the presence is duplicated in the pixel (i.e., if there was already another presence before at the same pixel) or FALSE if the presence is unique or the first record for a given pixel. 3. Loop over species 1. Identify the rows of *presence* table referring to the current species in the loop 2. Detect of duplicates only for the current species 3. Update the *duplicated* column created in step 2 with the relevant information for the species. 4. Print in the console how many duplicates were found for each species

```r
sps <- unique(pres$species)

pres$duplicated <- NA

for (sp in sps) {
    rows <- which(pres$species == sp)
    sp.dup <- duplicated(dt$cell[rows])
    pres$duplicated[rows] <- sp.dup
    print(paste("Species", sp, "has", sum(sp.dup), "duplicates"))
}
```

```
## [1] "Species Vaspis has 9349 duplicates"
## [1] "Species Vlatastei has 959 duplicates"
## [1] "Species Vseoanei has 750 duplicates"
```

We can check the first rows of data to check the table and how many presences were identified in total (each TRUE is equivalent to 1, and a FALSE to 0)

```r
head(pres)
```

```
##   species    x     y duplicated
## 1  Vaspis 2.70 42.05      FALSE
## 2  Vaspis 2.09 42.31      FALSE
## 3  Vaspis 2.70 41.87      FALSE
## 4  Vaspis 1.97 42.49      FALSE
## 5  Vaspis 2.09 42.13      FALSE
## 6  Vaspis 3.06 42.50      FALSE
```

```r
sum(pres$duplicated)
```

```
## [1] 11058
```

Everything seems coherent, so we proceed to remove the duplicated rows. We have to invert the logical value as we want to keep the rows that are not duplicated (were set as FALSE). The sub setting with [ ] only keeps the rows set as TRUE, thus, by inverting with the exclamation point (!) before, we invert and set TRUE to FALSE, and FALSE to TRUE.

```r
final_pres <- pres[!pres$duplicated, 1:3]
```

We only kept the first 3 columns (species, name, longitude and latitude) as they are the information we need to model.

We can get a summary of how may presence data points we kept at the end of presence processing.

```r
dim(final_pres)
```

```
## [1] 6796    3
```

```r
table(final_pres$species)
```

```
##
##    Vaspis Vlatastei  Vseoanei
##      4969      1227       600
```

Finally, we end this chapter by saving a new file with the optimized data set that will be used for modelling.

```r
# write to file
filename <- "data/species/speciesPresence_v2.csv"
write.table(final_pres, filename, sep="\t", row.names=FALSE, col.names=TRUE)
```

Chapter 4 - Variable selection for modelling

It is common practice to collect all predictor variables available that are deemed relevant for our study system and hypotheses. These include direct variables (like temperature) and processed variables (like distances to habitat categories). However, it is rarely a good idea to include all variables into the modeling process.

To address this, we can reduce the number of variables by checking the pairwise correlation among them. Correlated variables will bring mostly the same information to the model and might add a confounding effect when analyzing the variable importance in the models. There is no universal maximum correlation value for modeling, but avoiding correlations higher than 0.7 is generally considered sufficient. We will set this target for this example.

We will only need `terra` library for this.

```
library(terra)
```

We need the raster variables created in Chapter 2 and the optimised presence data from Chapter 3.

```
evi <- rast("data/rasters/evi.tif")
clim <- rast("data/rasters/climate.tif")
pres <- read.table("data/species/speciesPresence_v2.csv", sep="\t", header=TRUE)
```

The rasters are fully aligned (Chapter 2) and we can stack them together. Having the 20 variables (19 bioclimatic + 1 EVI) in the same object will simplify the process of selection.

```
rst <- c(clim, evi)
names(rst)
```

```
##  [1] "BIO_1"  "BIO_2"  "BIO_3"  "BIO_4"  "BIO_5"  "BIO_6"  "BIO_7"  "BIO_8"
##  [9] "BIO_9"  "BIO_10" "BIO_11" "BIO_12" "BIO_13" "BIO_14" "BIO_15" "BIO_16"
## [17] "BIO_17" "BIO_18" "BIO_19" "evi"
```

Define a training area with a buffer

We need to define a training area for the modeling. This restricts the available area from where we retrieve data for model calibration, including both presence data and pseudo-absence data. There are many ways to select this area, but we will follow a strategy that defines a spatial buffer around the presence points. This buffer limits the area from which we can retrieve environmental data from the rasters. A buffer should provide enough environmental variation for the modeling. However, a very wide buffer can pose problems, as we might include areas that are far from the species' optimal conditions. For instance, if we model a species only present in the Iberian Peninsula, but we choose a buffer large enough to include Iceland, we will be providing useless environmental variation to the model.

In this chapter, the buffer will define the training area that will provide environmental variation to the model and where we should check the correlations. Although we have more environmental data outside the buffer, that data will not be relevant for the model and hence should not be used for correlation analysis.

Here, we define the buffer size as 1 degree, which is roughly equivalent to ~110,000 meters.

NOTE: The buffer defined here will be important for Chapter 6 when we build the models. We will have to define the same buffer size."

NOTE on spatial data in R: When we open the presence data in R with the `read.table` command, we just create a data frame (like a spreadsheet in Excel). Although we might have columns for longitude and latitude, they are just numbers organized in two columns. Since creating buffers is a spatial operation, we have to inform R that the presences are spatial points. In other words, we have to formalize that the data frame is, in fact, a list of spatial points. This can be done with the `vect` command from the `terra` package, which is also used to open spatial data files such as shapefiles. We must define which columns store the longitude and latitude (in our case, the 'x' and 'y' columns).

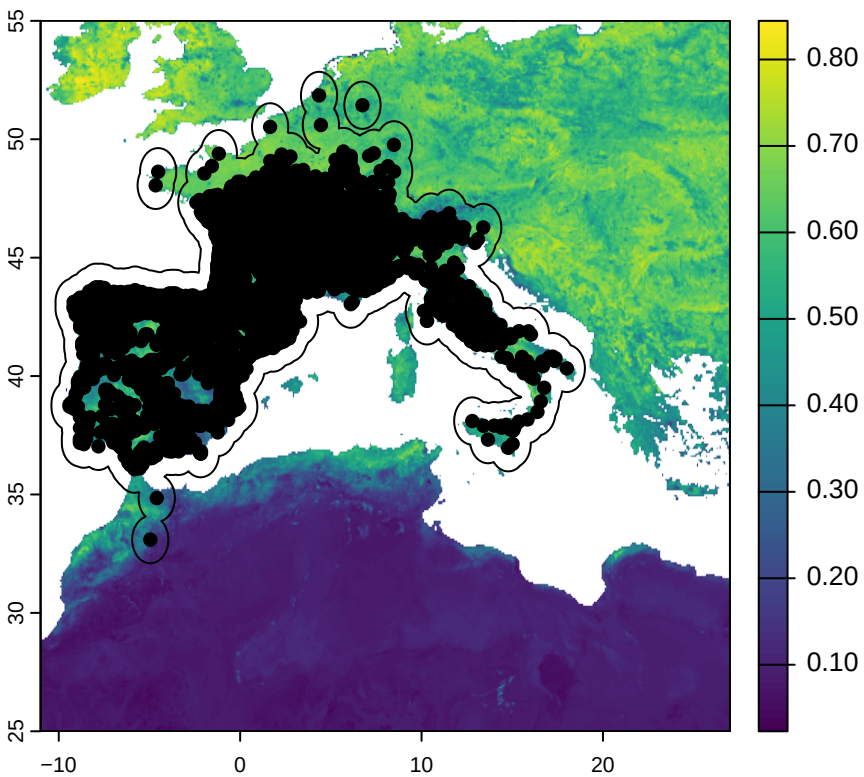
```
v <- vect(pres, geom= c("x", "y"))
```

Now we can define our buffers with a 1-degree radius. The `buffer` function creates an individual buffer around each presence, and we then `aggregate` them into a single polygon.

```
bsize <- 1
buf <- buffer(v, bsize)
buf <- aggregate(buf)
```

To check how everything is looking, we can plot presence and buffers over the EVI variable.

```
plot(evi)
plot(buf, cex=0.25, add=T)
plot(v, add=T)
```



As we can see in the figure, the buffer is created around presences. This includes ocean areas which will be ignored later as they provides only NAs.

We can use the buffer to extract the environmental data for all 20 variables.

```
dt <- extract(rst, buf, ID=FALSE)
head(dt)
```

```
##   BIO_1 BIO_2 BIO_3 BIO_4 BIO_5 BIO_6 BIO_7 BIO_8 BIO_9 BIO_10 BIO_11 BIO_12
## 1    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
## 2    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
## 3    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
## 4    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
## 5    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
## 6    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA     NA     NA
##   BIO_13 BIO_14 BIO_15 BIO_16 BIO_17 BIO_18 BIO_19 evi
## 1     NA     NA     NA     NA     NA     NA     NA  NA
## 2     NA     NA     NA     NA     NA     NA     NA  NA
## 3     NA     NA     NA     NA     NA     NA     NA  NA
## 4     NA     NA     NA     NA     NA     NA     NA  NA
## 5     NA     NA     NA     NA     NA     NA     NA  NA
## 6     NA     NA     NA     NA     NA     NA     NA  NA
```

There are many NAs that come from ocean pixels. We should remove those rows and keep only the valid data before proceeding with correlations. Since the rasters are fully aligned (Chapter 2), including the No Data mask, we can use the NAs from a single raster to filter all other variables.

```
dt <- dt[!is.na(dt[,1]),]
head(dt)
```

```
##       BIO_1    BIO_2    BIO_3    BIO_4    BIO_5     BIO_6    BIO_7    BIO_8
## 8  9.446667 5.786667 29.61447 509.8863 20.14000 0.6000000 19.54000 10.74000
## 18 9.408896 5.943919 30.25103 508.3113 20.18378 0.5351351 19.64865 10.68108
## 19 9.499875 6.132250 30.78748 512.5685 20.44500 0.5270000 19.91800 10.69300
## 20 9.612041 6.353250 31.41441 520.0265 20.71900 0.4950000 20.22400 10.71300
## 31 9.459243 6.206061 31.10382 510.2022 20.35273 0.4000000 19.95273 10.69364
## 32 9.627584 6.378666 31.63864 514.4630 20.68200 0.5210000 20.16100 10.72333
##        BIO_9   BIO_10   BIO_11 BIO_12 BIO_13 BIO_14   BIO_15 BIO_16 BIO_17
## 8   5.166667 15.74667 3.450000    798     99     40 30.06747    287    143
## 18  5.153153 15.66441 3.411712    799     99     41 30.21513    288    143
## 19  5.267500 15.77517 3.431667    803     98     41 29.25621    286    144
## 20  5.363167 15.93750 3.436833    825     98     43 27.84145    288    148
## 31 11.040606 15.70848 3.421818    818    101     42 30.15000    293    146
## 32  5.438000 15.90683 3.519333    813    101     41 29.37465    289    147
##    BIO_18 BIO_19       evi
## 8     212    192 0.5183802
## 18    211    192 0.4421933
## 19    214    192 0.6296464
## 20    190    195 0.5911894
## 31    213    198 0.4945980
## 32    182    195 0.6150222
```

Now we have only data! We can proceed with Pearson correlations.

Calculate pairwise correlations

Getting the Pearson's correlation scores for each pair of variables is very easy in R. We have all raster data for the training area (buffer) organized in a data frame where each column is a different variable. We just calculate correlations based on that table.

```
corr <- cor(dt)
# Rounding to 3 decimal places to facilitate visualization.
round(corr, 3)
```

```
##         BIO_1  BIO_2  BIO_3  BIO_4  BIO_5  BIO_6  BIO_7  BIO_8  BIO_9 BIO_10
```

```
## BIO_1   1.000  0.429  0.516 -0.087  0.870  0.896  0.217  0.410  0.737  0.956
## BIO_2   0.429  1.000  0.675  0.433  0.767  0.073  0.855  0.129  0.422  0.566
## BIO_3   0.516  0.675  1.000 -0.345  0.520  0.458  0.202  0.020  0.538  0.420
## BIO_4  -0.087  0.433 -0.345  1.000  0.325 -0.476  0.831  0.201 -0.181  0.207
## BIO_5   0.870  0.767  0.520  0.325  1.000  0.585  0.661  0.360  0.676  0.957
## BIO_6   0.896  0.073  0.458 -0.476  0.585  1.000 -0.222  0.275  0.690  0.737
## BIO_7   0.217  0.855  0.202  0.831  0.661 -0.222  1.000  0.178  0.174  0.468
## BIO_8   0.410  0.129  0.020  0.201  0.360  0.275  0.178  1.000 -0.046  0.448
## BIO_9   0.737  0.422  0.538 -0.181  0.676  0.690  0.174 -0.046  1.000  0.685
## BIO_10  0.956  0.566  0.420  0.207  0.957  0.737  0.468  0.448  0.685  1.000
## BIO_11  0.962  0.304  0.587 -0.349  0.739  0.967 -0.006  0.314  0.764  0.844
## BIO_12 -0.514 -0.489 -0.263 -0.240 -0.630 -0.340 -0.443 -0.319 -0.382 -0.583
## BIO_13 -0.242 -0.348 -0.096 -0.292 -0.387 -0.100 -0.373 -0.219 -0.131 -0.326
## BIO_14 -0.734 -0.544 -0.472 -0.034 -0.766 -0.585 -0.380 -0.256 -0.678 -0.746
## BIO_15  0.627  0.385  0.431 -0.118  0.583  0.539  0.201  0.134  0.609  0.595
## BIO_16 -0.272 -0.375 -0.101 -0.322 -0.425 -0.112 -0.407 -0.251 -0.155 -0.365
## BIO_17 -0.719 -0.530 -0.468 -0.019 -0.747 -0.582 -0.359 -0.270 -0.636 -0.725
## BIO_18 -0.751 -0.511 -0.512  0.059 -0.760 -0.658 -0.304 -0.069 -0.777 -0.736
## BIO_19 -0.100 -0.301  0.103 -0.497 -0.293  0.114 -0.458 -0.426  0.097 -0.242
## evi    -0.230 -0.487 -0.269 -0.193 -0.392 -0.039 -0.435 -0.055 -0.344 -0.302
##        BIO_11 BIO_12 BIO_13 BIO_14 BIO_15 BIO_16 BIO_17 BIO_18 BIO_19    evi
## BIO_1   0.962 -0.514 -0.242 -0.734  0.627 -0.272 -0.719 -0.751 -0.100 -0.230
## BIO_2   0.304 -0.489 -0.348 -0.544  0.385 -0.375 -0.530 -0.511 -0.301 -0.487
## BIO_3   0.587 -0.263 -0.096 -0.472  0.431 -0.101 -0.468 -0.512  0.103 -0.269
## BIO_4  -0.349 -0.240 -0.292 -0.034 -0.118 -0.322 -0.019  0.059 -0.497 -0.193
## BIO_5   0.739 -0.630 -0.387 -0.766  0.583 -0.425 -0.747 -0.760 -0.293 -0.392
## BIO_6   0.967 -0.340 -0.100 -0.585  0.539 -0.112 -0.582 -0.658  0.114 -0.039
## BIO_7  -0.006 -0.443 -0.373 -0.380  0.201 -0.407 -0.359 -0.304 -0.458 -0.435
## BIO_8   0.314 -0.319 -0.219 -0.256  0.134 -0.251 -0.270 -0.069 -0.426 -0.055
## BIO_9   0.764 -0.382 -0.131 -0.678  0.609 -0.155 -0.636 -0.777  0.097 -0.344
## BIO_10  0.844 -0.583 -0.326 -0.746  0.595 -0.365 -0.725 -0.736 -0.242 -0.302
## BIO_11  1.000 -0.430 -0.154 -0.701  0.640 -0.174 -0.689 -0.742  0.039 -0.192
## BIO_12 -0.430  1.000  0.899  0.714 -0.251  0.915  0.773  0.741  0.784  0.369
## BIO_13 -0.154  0.899  1.000  0.371  0.151  0.990  0.453  0.480  0.864  0.182
## BIO_14 -0.701  0.714  0.371  1.000 -0.800  0.396  0.984  0.891  0.255  0.541
## BIO_15  0.640 -0.251  0.151 -0.800  1.000  0.133 -0.766 -0.617  0.186 -0.501
## BIO_16 -0.174  0.915  0.990  0.396  0.133  1.000  0.468  0.502  0.885  0.196
## BIO_17 -0.689  0.773  0.453  0.984 -0.766  0.468  1.000  0.891  0.318  0.515
## BIO_18 -0.742  0.741  0.480  0.891 -0.617  0.502  0.891  1.000  0.204  0.423
## BIO_19  0.039  0.784  0.864  0.255  0.186  0.885  0.318  0.204  1.000  0.177
## evi    -0.192  0.369  0.182  0.541 -0.501  0.196  0.515  0.423  0.177  1.000
```

The correlation matrix provides the correlation score for each pair of variables. It is symmetric (the correlation of A to B is the same as B to A) and the diagonal gives the self-correlation, which is obviously 1.

This matrix is not easy to use for the variable elimination process. Remember that we set a goal of not having an absolute correlation higher than 0.7, so we should find a way to select important variables that will provide a new set where the maximum pairwise correlation won't exceed that value.

One of the best and simplest ways is to organize the variables in a dendrogram by using a hierarchical clustering method. We should group in the same cluster variables that are similar to each other (highly correlated). To do this, we need to transform the correlation score into a dissimilarity or distance matrix, where increasing values will reflect that variables are less similar (low correlation). For that we need to:

1. Get the absolute value of the correlation because we just care about the magnitude of the correlation and not the direction (two variables that are correlated at -0.87 provide the same information as those

that are correlated at +0.87). At this point, the matrix will have values from 0 (low correlation, thus highly distant) to 1 (high correlation, showing high similarity and low distance).
2. Ensure that lower values reflect similarity (high correlation) and higher values describe dissimilarity (low correlation). To achieve this, we need to invert the values by subtracting the absolute value from 1.

```
# Convert correlation to distance
dist <- 1 - abs(corr)
```

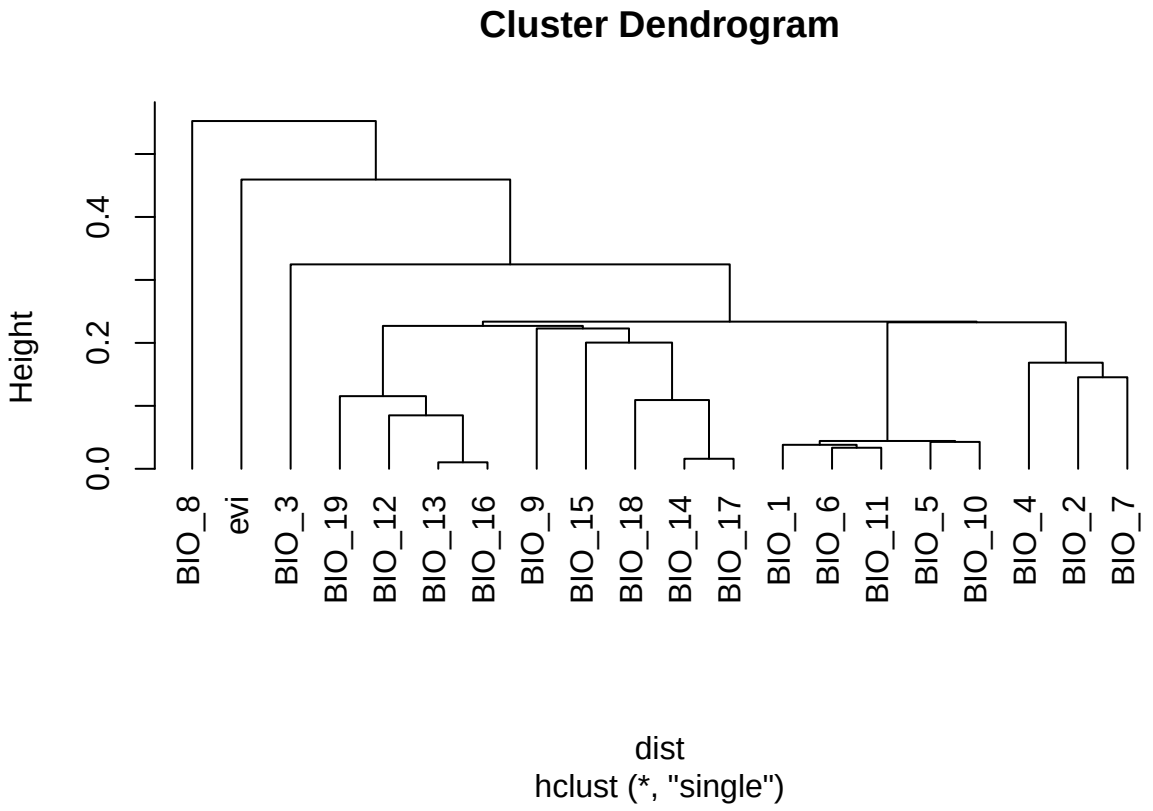
We need to discard one of the symmetrical sides and the diagonal. An easy way to do this is to convert the matrix to a `dist` object:

```
dist <- as.dist(dist)
```

Clustering and selection

With the new distance matrix we can start the clustering process. We choose the method 'single' so every cluster is grouped by the minimum distance/correlation.

```
# Hierarchical cluster of distance
hc <- hclust(dist, method="single")
plot(hc, hang=-1)
```



We can select 5 variables, one from each major group in the tree. Variables can be selected based on how easily they can be interpreted in terms of the niche of the species. In this case, we select:

- evi
- BIO_8: Mean daily Temperature Wettest Quarter
- BIO_3: Isothermality
- BIO_12: Annual Precipitation Amount
- BIO_1: Annual Temperature

We select the vars using the variables names we defined previously and create a new raster with only those variables.

```
sel_vars <- c("evi", "BIO_8", "BIO_3", "BIO_12", "BIO_1")
sel_rst <- rst[[sel_vars]]
```

We select the variables using the names we defined previously and create a new raster with only those variables.

```
# Check correlation of final dataset
sel_dt <- dt[,sel_vars]
cor(sel_dt)
```

```
##                evi       BIO_8       BIO_3     BIO_12      BIO_1
## evi     1.00000000 -0.05473548 -0.26896220  0.3687846 -0.2295176
## BIO_8  -0.05473548  1.00000000  0.01993914 -0.3192758  0.4101052
## BIO_3  -0.26896220  0.01993914  1.00000000 -0.2634080  0.5157017
## BIO_12  0.36878463 -0.31927583 -0.26340797  1.0000000 -0.5142887
## BIO_1  -0.22951757  0.41010518  0.51570173 -0.5142887  1.0000000
```

The maximum correlation does not pass the value of 0.51, which is generally acceptable.

We save the selected variables to a file named *final_vars.tif*. You can check the variables in any GIS.

```
# Write Final raster to model
writeRaster(sel_rst, "data/rasters/final_vars.tif", overwrite=TRUE)
```

Chapter 5 - Prepare projection variables

Our initial question is about the contact zone between three species of vipers in the Iberian Peninsula. Although the original distributions of two of the species, *Vipera aspis* and *Vipera latastei* extend beyond the Iberian Peninsula, we don't need the final projections to cover the full extent of their distributions. To answer our question, we only need predictions for the Iberian Peninsula.

In this chapter, we will prepare the variables for model projection. This includes current climate variables and NDVI, as well as future climate predictions for the same selected variables used in model training (Chapter 4). The projection area is defined as the Iberian Peninsula. We will crop and mask the variables to show data only for the Iberian Peninsula.

A mandatory detail is that all groups of projection variables must have the same layer names as those used in model training so that the models can find the respective variables to produce predictions.

The names we have are:

- evi
- BIO_8
- BIO_3
- BIO_12
- BIO_1

Keep in mind that NDVI is a static variable in this scheme, meaning we don't have future projections of change for NDVI, so we use the same variable for all datasets. This approach is not usually done, as NDVI is very susceptible to changes due to climate change and human activity. Therefore, we are assuming that the habitat, for which we use NDVI as a surrogate, is stable over time. Variables like elevation, or derived variables like slope, are more easily assumed to be static.

Again, we only need `terra` library for this chapter, and `geodata` to download future climate layers at the end of the chapter.

```
library(terra)
library(geodata)
```

To crop and cut the rasters, we use the Iberian Peninsula shapefile

```
# Open Iberia shape file
v <- vect("data/other/iberia.shp")
```

Prepare current dataset for projection

We open the current data set that we prepare in chapter 4.

```
# Open current data rasters
vars <- rast("data/rasters/final_vars.tif")
```

We use the Iberian Peninsula shapefile to crop and mask the rasters. Cropping means reducing the extent, while masking is used to define No Data for all pixels outside the Iberian Peninsula.

```
# Cut Current data and save
vars <- crop(vars, v, mask=TRUE)
```

The current dataset for projection is ready, and we can write it to a file. All projection files will be saved in the *data/rasters* folder, and filenames will start with a "proj_" prefix.

```
writeRaster(vars, "data/rasters/proj_current.tif", overwrite=TRUE)
```

Prepare future climate projections

Climate projections add extra complexity because there are many variants. As it is difficult to predict future climate, Global Circulation Models (GCMs) are given different scenarios for possible paths of societal behavior towards greenhouse gas production. These scenarios are called the Shared Socioeconomic Pathways (SSPs) and range from maintaining the current trend of emissions (worst-case scenario) to maintaining and even reducing emissions (best-case scenario). This allows us to predict climate over a range of conditions that we should consider in our niche model projections. Therefore, we can project niche models to different GCMs, different SSPs, and different time periods.

In this tutoral we will project the models to * One single GCM (mpi-esm1-2-hr) * Two SSPS (ssp126 and ssp585) * Three time periods (2011-2040, 2041-2070, 2071-2100)

This result in a combination of 1x2x3=6 model projections.

All *raw* future climate rasters will be saved in the folder *data/rasters/original/climate*.

The easiest way to process all data is to loop over AGES and SSPs to produce all combinations of projections. This will allow to crop and mask all needed projections variables but it requires a nested *for* loop.

First we can set some objects with needed information:

```
ages <- c("2021-2040", "2041-2060", "2061-2080")
scenarios <- c("126", "585")

# bioclimatic in the same order as in vars to simplify
bios <- c("BIO_8", "BIO_3", "BIO_12", "BIO_1")
```

We can now loop over ages and ssp scenarios. The process inside the nested loop is:

1. DowNload the dataset for the combination of GCM model + age + ssp (~10km spatial resolution) using `geodata` and writing to *data/rasters/original*.
2. Crop the rasters to the study area (same extent as `vars`).
3. Rename layers to follow our name convention.

4. Select only the variables of interest from the 19 bioclimate variables.
5. Join the evi (static) to the set of selected bioclimate variables
6. Write the produced raster with the 4 layers in the *data/rasters/* folder and following a filename convention that starts by *proj_age_ssp* in `TIF` format.

```
for (age in ages) {
    for (ssp in scenarios) {
        # download and rename
        clim_fut <- cmip6_world(model='MPI-ESM1-2-HR', ssp=ssp, time=age, var='bioc', res=5, path='data/
        clim_fut <- crop(clim_fut, vars[[1]], mask=TRUE)
        names(clim_fut) <- paste0("BIO_", 1:19)
        #filter the selected vars
        clim_fut <- clim_fut[[bios]]
        # merge with evi
        fut <- c(vars[["evi"]], clim_fut)
        # Write raster to file
        writeRaster(fut, paste0("data/rasters/proj_", age, "_", ssp, ".tif"), overwrite=TRUE)
    }
}
```

At the end of this loop, we should have 6 projection rasters for future time periods plus 1 projection for current period.

Chapter 6 - Model building

Until now, we have been preparing data for Ecological Niche Modelling (ENM). These included_

1. Clean presence locations
2. Aligned and selected environmental variables for training
3. Cropped projection rasters for present and future climates

At this point, we are ready to produce the first models. We will be using a package named *biomod2* that provides functionality for modeling, ensembling multiple models, and retrieving model projections.

Building a single model is straightforward. However, if we want to vary some initial conditions (resampling, pseudo-absence selection, etc.) and use multiple modeling algorithms (GLM, GAM, CART, Maxent, etc.), we need to produce a large number of models and predictions. The *biomod2* library facilitates this process by internally tracking all model variants and providing us with a simpler interface to control crucial parameters of the modeling process that require our intervention.

The purpose of this chapter is to build an ENM with:

- 3 sets of pseudo-absences drawn randomly from the buffer area around presences
- 4 modeling algorithms including Generalised Linear Models (GLM), General Additive Models (GAM), Maximum Entropy (Maxnet), and eXtreme Gradient Boosting Training (XGBOOST)
- 5 replicates using K-Fold resampling strategy.

The combination results in $3x4x5 = 60$ build models. Besides those, we will have to project models to present conditions and 6 future climate layers from a combination of ages and SSPs (covered in chapter 8).

"We need the *terra* and *biomod2* packages. While *terra* provides GIS capabilities, *biomod2* offers the tools to build models. If *biomod2* is not available, it requires installation. Additionally, *biomod2* depends on other libraries, including *dismo*, *xgboost*, *maxnet*, *gam*, and *tidyterra*. We don't need to interact directly with these packages, but *biomod2* requires them. We can install these packages either via the RStudio menu or by executing the following code:

```
install.packages("biomod2")
install.packages("dismo")
```

```
install.packages("xgboost")
install.packages("maxnet")
install.packages("gam")
install.packages("tidyterra")
```

And we can open the need libraries:

```
library(terra)
library(biomod2)
```

Open the relevant data for modelling.

```
pres <- read.table("data/species/speciesPresence_v2.csv", sep="\t", header=TRUE)
vars <- rast("data/rasters/final_vars.tif")
```

Recall that the presence table has data for the 3 species and that the selected variables are in a single raster file.

Model building

In this chapter, we will build models for three species. We'll start with a detailed walkthrough for *Vipera aspis*, and then apply the same process to the other two species. All models will be saved in the models directory, which should be empty at the start. A folder for each species, organized by *biomod2*, will be created automatically.

The first step is to provide data into a formatting functions that will prepare the data in a way that *biomod2* can understand and track parameters.

We need a response variable which is a vector of 1s with the same size as presence locations for the species. This is because we don't have absences (zeros) and we are using a presence-only model strategy by creating pseudo-absences.

We also need a table of coordinates for each presence which is present in our presence data file.

Finally we need our explanatory variables or predictors that we prepared before.

```
# How many presences for Vaspis?
n_va <- sum(pres$species == "Vaspis")

# repeat 1 to the same length of presences available
resp_va <- rep(1, n_va)

# get the coordinates for Vaspis
coords_va <- pres[pres$species == "Vaspis", 2:3]
```

In the formatting data, we also provide some other modeling parameters. We need to define the strategy for selecting pseudo-absences, which is based on a buffer around presences. We use the same buffer as before (Chapter 2), which is 110,000 km. We ask for three different sets of 10,000 pseudo-absences followinf the **disk** strategy (same as buffer), which will allow us to capture some variation in the initial conditions for each model.

The *dir.name* sets the folder where models are to be saved and the *resp.name* sets a name for the project and folder to be created inside the *models* folder.

```
vaData <- BIOMOD_FormatingData(resp.var = resp_va,
                               expl.var = vars,
                               resp.xy = coords_va,
                               resp.name = "Vaspis",
                               dir.name = "models",
```

```
                                PA.nb.rep = 3,
                                PA.nb.absences = 10000,
                                PA.strategy = "disk",
                                PA.dist.max = 110000)
```

```
##
## -=-=-=-=-=-=-=-=-=-=-=-=-= Vaspis Data Formating -=-=-=-=-=-=-=-=-=-=-=-=-=
##
##       ! No data has been set aside for modeling evaluation
##       ! No data has been set aside for modeling evaluation
##
## Checking Pseudo-absence selection arguments...
##
##       ! No data has been set aside for modeling evaluation
##    > Disk pseudo absences selection
##    > Pseudo absences are selected in explanatory variables|---------|---------|---------|---------|=
##    > random pseudo absences selection
##    > Pseudo absences are selected in explanatory variables
##
##       ! No data has been set aside for modeling evaluation
##       ! No data has been set aside for modeling evaluation
## -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Done -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

We have the data formatted, so we can proceed to the modeling step. Here, we need to define the model algorithms to run and the train/test strategy. This allows us to control overfitting by providing examples to train the model and a set of independent examples to test its predictive ability. We choose K-fold cross-validation for this purpose. In this case, we divide the dataset into five equal-sized folds and test four against one for each fold.

As models should be evaluated for performance, we choose two common techniques for this: TSS (True Skill Statistic) and ROC (Receiver Operating Characteristic)

There are many algorithms available for modelling. We are limiting to 4 different algorithms which are generally fast to train and still provide a good example of the ensemble modelling strategy. We choose:

- Generalized Linear Model (GLM): A general regression model using a binomial family of distributions (logistic regression)
- Generalized Additive Models (GAM): A regression-based approach where smoothing functions are applied to each predictor.
- Maximum Entropy (Maxnet): A machine learning method sharing some similarities with GAM and is an R version of the common MaxEnt approach.
- Extreme Gradient Boosting (XGBOOST): A machine learning method related to decision trees, known to be fast and perform well.

Each of these algorithms has specific parameters to tune. The *biomod2* library offers different strategies for tuning them. We are choosing the 'bigboss' strategy, which consists of a list of the best parameters as defined by the package authors. This means that the authors tested several algorithms under different data and provided a list of options for each algorithm that generally performs well in most common situations.

Variable importance is a measure of how much each predictor influences each model. In the context of Ecological Niche Modeling (ENM), it detects the variable most determining the distribution of the species. This is done via permutations, set in *var.import*. We set this to 3, which is a low number of permutations, but since they take some time to run, we need to keep this number low for this example.

```
vaModel <- BIOMOD_Modeling(bm.format = vaData,
                           modeling.id = "EcoMod",
                           models = c("GAM", "GLM", "MAXNET", "XGBOOST"),
```

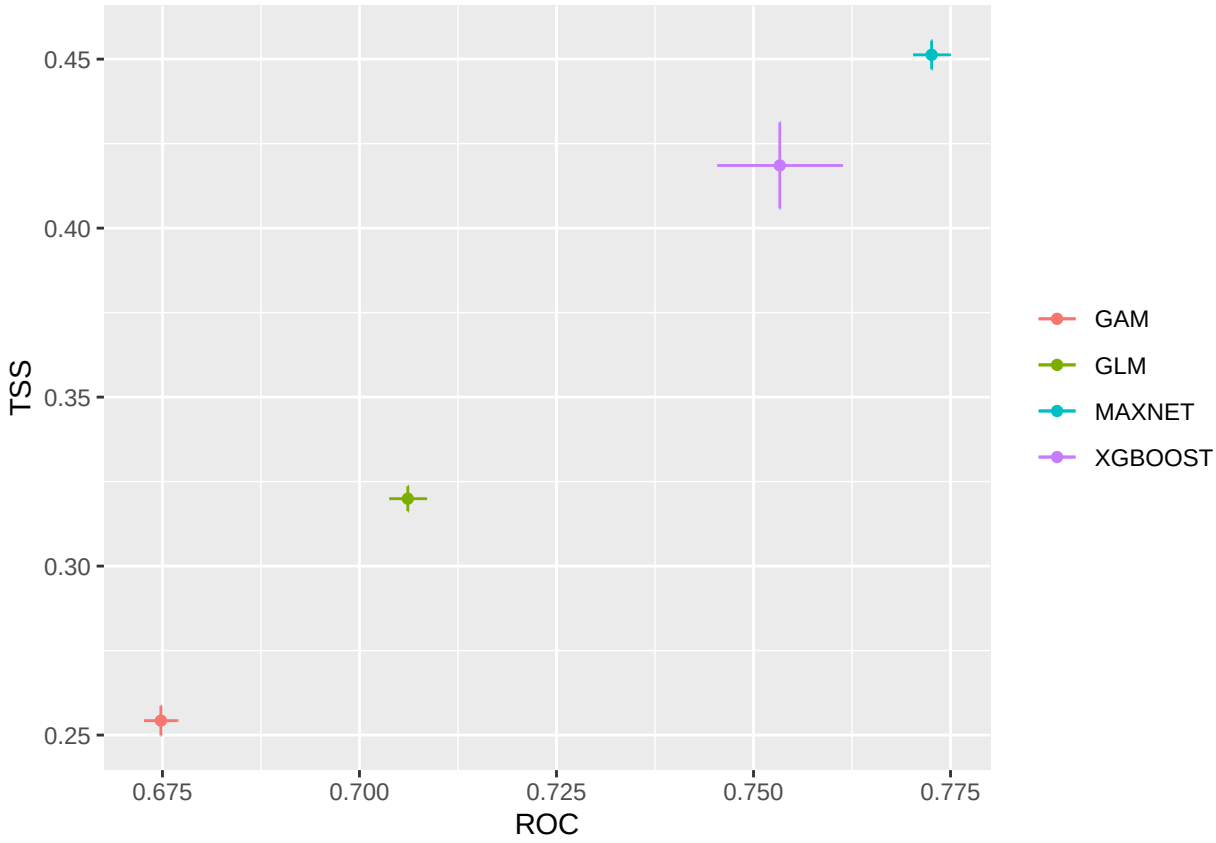
```
                            CV.strategy = "kfold",
                            CV.k = 5,
                            CV.do.full.models = FALSE,
                            OPT.strategy = "bigboss",
                            var.import = 3,
                            metric.eval = c("TSS", "ROC"))
```

```
vaModel
```

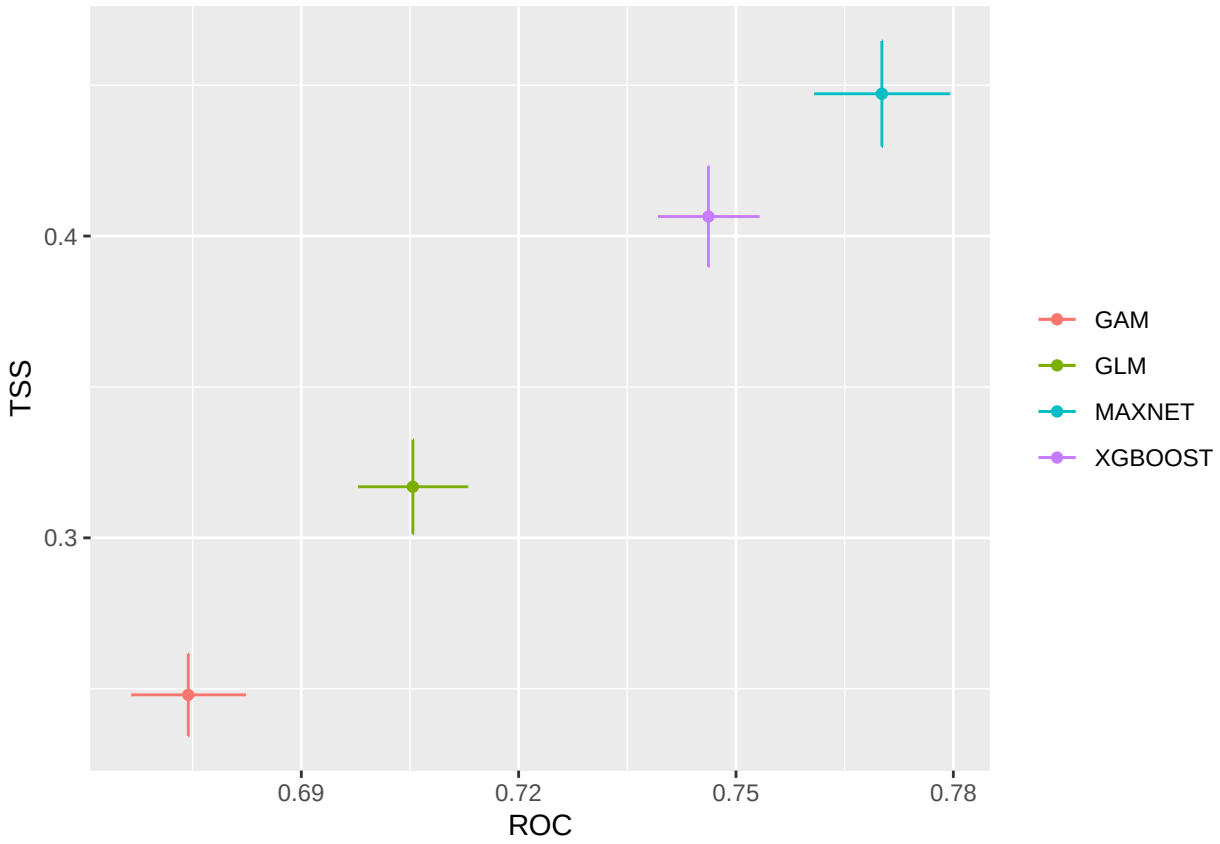
```
##
## -=-=-=-=-=-=-=-=-=-=-=-=-=-= BIOMOD.models.out -=-=-=-=-=-=-=-=-=-=-=-=-=-=
##
## Modeling folder : models
##
## Species modeled : Vaspis
##
## Modeling id : EcoMod
##
## Considered variables : evi BIO_8 BIO_3 BIO_12 BIO_1
##
##
## Computed Models :  Vaspis_PA1_RUN1_GAM Vaspis_PA1_RUN1_GLM
## Vaspis_PA1_RUN1_MAXNET Vaspis_PA1_RUN1_XGBOOST Vaspis_PA1_RUN2_GAM
## Vaspis_PA1_RUN2_GLM Vaspis_PA1_RUN2_MAXNET Vaspis_PA1_RUN2_XGBOOST
## Vaspis_PA1_RUN3_GAM Vaspis_PA1_RUN3_GLM Vaspis_PA1_RUN3_MAXNET
## Vaspis_PA1_RUN3_XGBOOST Vaspis_PA1_RUN4_GAM Vaspis_PA1_RUN4_GLM
## Vaspis_PA1_RUN4_MAXNET Vaspis_PA1_RUN4_XGBOOST Vaspis_PA1_RUN5_GAM
## Vaspis_PA1_RUN5_GLM Vaspis_PA1_RUN5_MAXNET Vaspis_PA1_RUN5_XGBOOST
## Vaspis_PA2_RUN1_GAM Vaspis_PA2_RUN1_GLM Vaspis_PA2_RUN1_MAXNET
## Vaspis_PA2_RUN1_XGBOOST Vaspis_PA2_RUN2_GAM Vaspis_PA2_RUN2_GLM
## Vaspis_PA2_RUN2_MAXNET Vaspis_PA2_RUN2_XGBOOST Vaspis_PA2_RUN3_GAM
## Vaspis_PA2_RUN3_GLM Vaspis_PA2_RUN3_MAXNET Vaspis_PA2_RUN3_XGBOOST
## Vaspis_PA2_RUN4_GAM Vaspis_PA2_RUN4_GLM Vaspis_PA2_RUN4_MAXNET
## Vaspis_PA2_RUN4_XGBOOST Vaspis_PA2_RUN5_GAM Vaspis_PA2_RUN5_GLM
## Vaspis_PA2_RUN5_MAXNET Vaspis_PA2_RUN5_XGBOOST Vaspis_PA3_RUN1_GAM
## Vaspis_PA3_RUN1_GLM Vaspis_PA3_RUN1_MAXNET Vaspis_PA3_RUN1_XGBOOST
## Vaspis_PA3_RUN2_GAM Vaspis_PA3_RUN2_GLM Vaspis_PA3_RUN2_MAXNET
## Vaspis_PA3_RUN2_XGBOOST Vaspis_PA3_RUN3_GAM Vaspis_PA3_RUN3_GLM
## Vaspis_PA3_RUN3_MAXNET Vaspis_PA3_RUN3_XGBOOST Vaspis_PA3_RUN4_GAM
## Vaspis_PA3_RUN4_GLM Vaspis_PA3_RUN4_MAXNET Vaspis_PA3_RUN4_XGBOOST
## Vaspis_PA3_RUN5_GAM Vaspis_PA3_RUN5_GLM Vaspis_PA3_RUN5_MAXNET
## Vaspis_PA3_RUN5_XGBOOST
##
##
## Failed Models :  none
##
## -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

This command will take some time to run as it needs to build all the combinations of models we specified. It displays general information on the process, allowing us to check if any models are failing. If models do fail, we may need to investigate the reasons and adjust some parameters. If everything runs successfully, we can plot the results to evaluate the models' performance.

```
plt <- bm_PlotEvalMean(vaModel, dataset="calibration")
```




```
plt <- bm_PlotEvalMean(vaModel, dataset="validation")
```



NOTE: We are assigning the results of the functions to a *plt* object that gets replaced with each plot. Since the functions used to plot also return the data used to build the plots, assigning the results to an object prevents displaying the entire dataset in the console. Although we are not using this data here, you can check the contents of *plt* at any time!

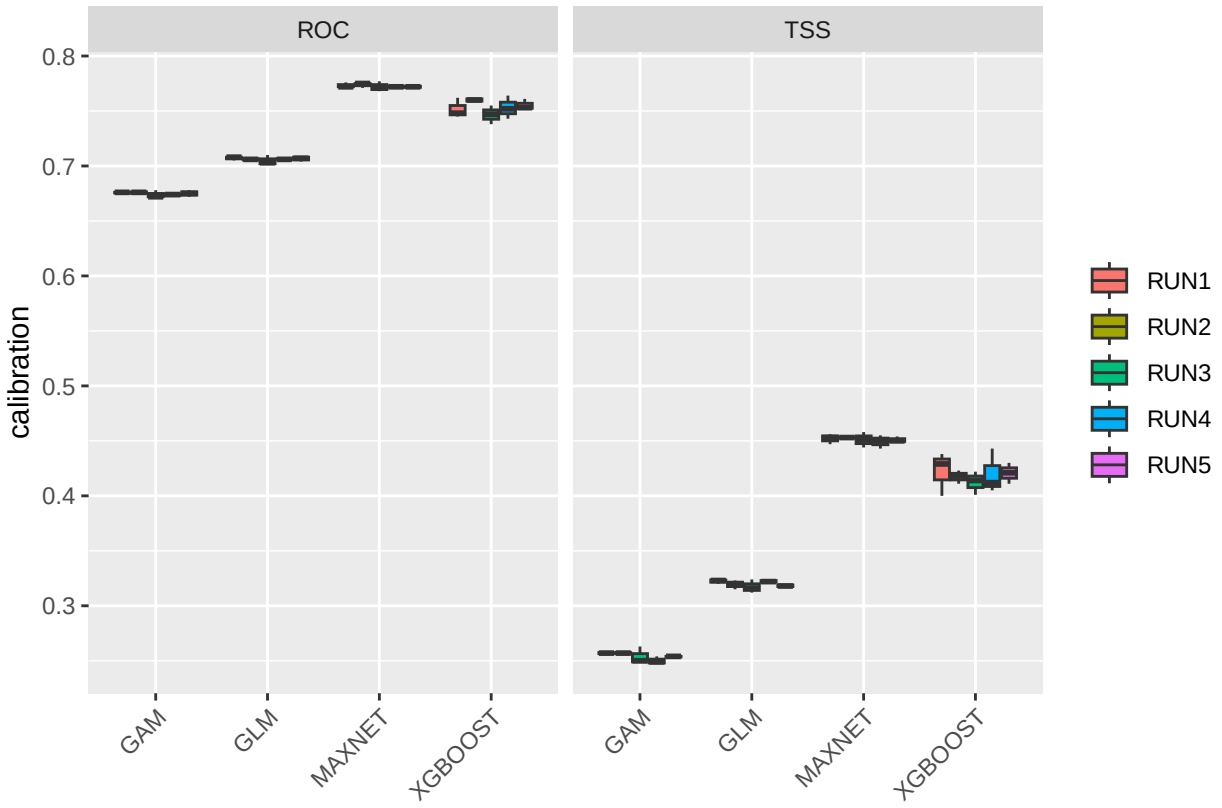
These plots show the model performance for the calibration and validation sets. Remember that we used the K-fold strategy with K=5K=5 folds. This means that all five combinations of four folds against one fold were used for calibration (training) and validation, respectively.

The plots show the TSS (True Skill Statistic) and the ROC/AUC (Receiver Operating Characteristic/Area Under Curve) values. The closer these values are to 1, the better the model performance. Of course, different algorithms, different sets of pseudo-absences, and different sets of training data (folds) will provide slight variations in performance. Generally, the models we built are acceptable, as indicated by ROC/AUC values higher than 0.65 and TSS greater than 0.25. In general, XGBoost and Maxnet, both machine learning methods, outperform the simpler regression-based methods.

Both calibration and validation show acceptable performance, indicating good model fitting, predictive ability, and a low level of overfitting.

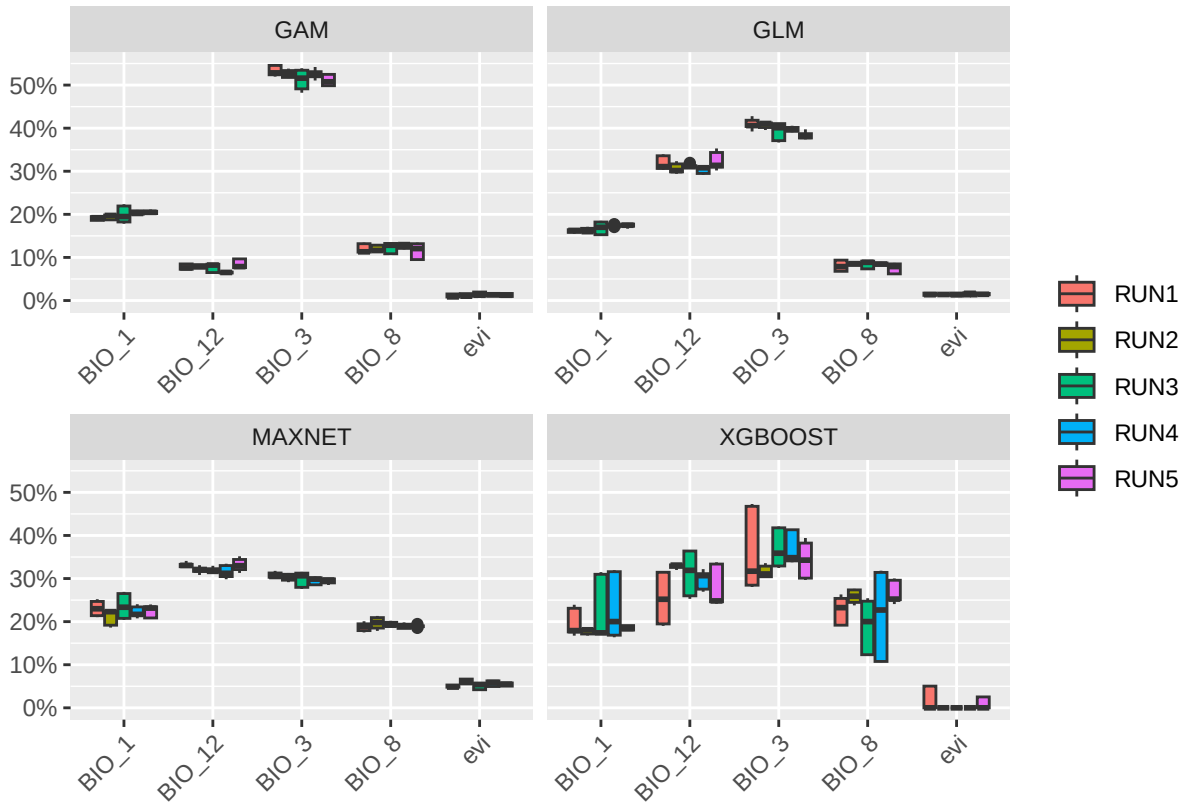
We can compare together the different runs and algorithms:

```
plt <- bm_PlotEvalBoxplot(vaModel, group.by = c('algo', 'run'))
```



An important step in the modeling results analysis is to understand how much each predictor variable is contributing to each model.

```
plt <- bm_PlotVarImpBoxplot(vaModel, group.by = c('expl.var', 'run', 'algo'))
```



In general, it is shown that EVI is a low contributor for this species, while climate variables are generally more associated with the distribution of V. aspis. Although precipitation (BIO_12) is associated with high presence for Maxnet and GLM, a temperature variable (BIO_3) is generally important to all models.

We can plot the response curves for each variable, which show how the predictions of the model vary with the range of values for each variable. This involves using the models to predict across the entire range of values for one variable while keeping the other variables fixed at a constant value. In this example, the other variables are fixed at their median value.

```
plt <- bm_PlotResponseCurves(vaModel, fixed.var = 'median')
```

There are many combinations of models, runs, and pseudo-absence sets, making these plots complex to interpret. After ensembling (chapter 7), we will construct simpler response curves. However, it is important to note that, in general, the responses show similar patterns. We should give more importance to the responses of the variables that are more significant for each algorithm. For instance, it is not useful tssp in sspso focus on EVI for Maxnet and GLM algorithms.

The models were built in the environmental space. The environmental values for the four predictors were extracted from the locations (presence and pseudo-absences) by *biomod2*. The models were then built, tested, and evaluated.

At this point, we have a large combination of models that will be ensembled into a single one in the next chapter. However, we can still project the model to the entire study area. This will create a raster file that we can open in any GIS, which is useful for checking the spatial prediction patterns for each individual model and perform visual checking. The projections we will use to answer our initial question will be made in chapter 8, only after ensembling the models.

```
vaProj <- BIOMOD_Projection(bm.mod = vaModel,
                            proj.name = 'Present',
                            new.env = vars,
                            models.chosen = 'all')
```

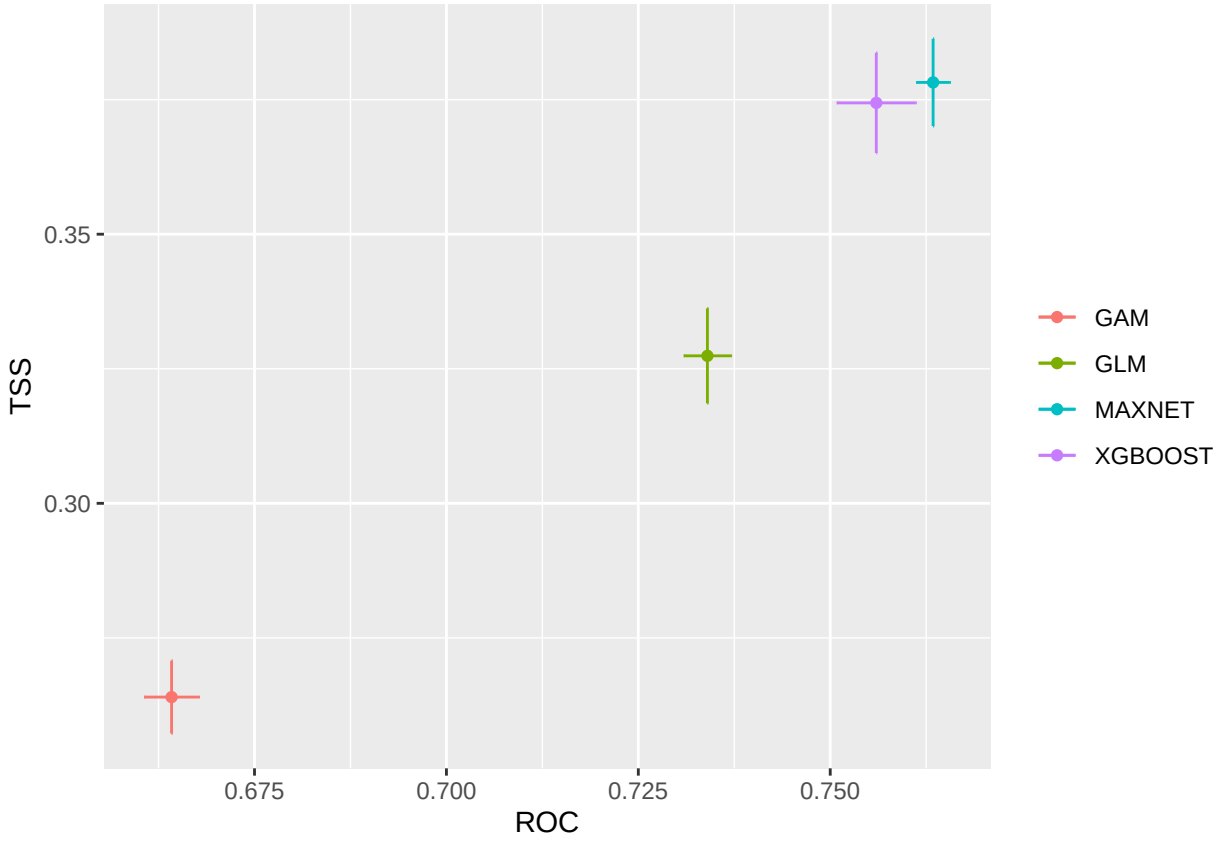
We can directly plot all models, but it might take a long time to finish since there are many models (PA x Runs x Algorithm = 3 x 5 x 4 = 60 models) to show. However, all models are built and saved into a single file, *models/Vaspis/proj_Present/proj_Present_Vaspis.tif*, which can be inspected in any GIS.

Repeat for the other two species

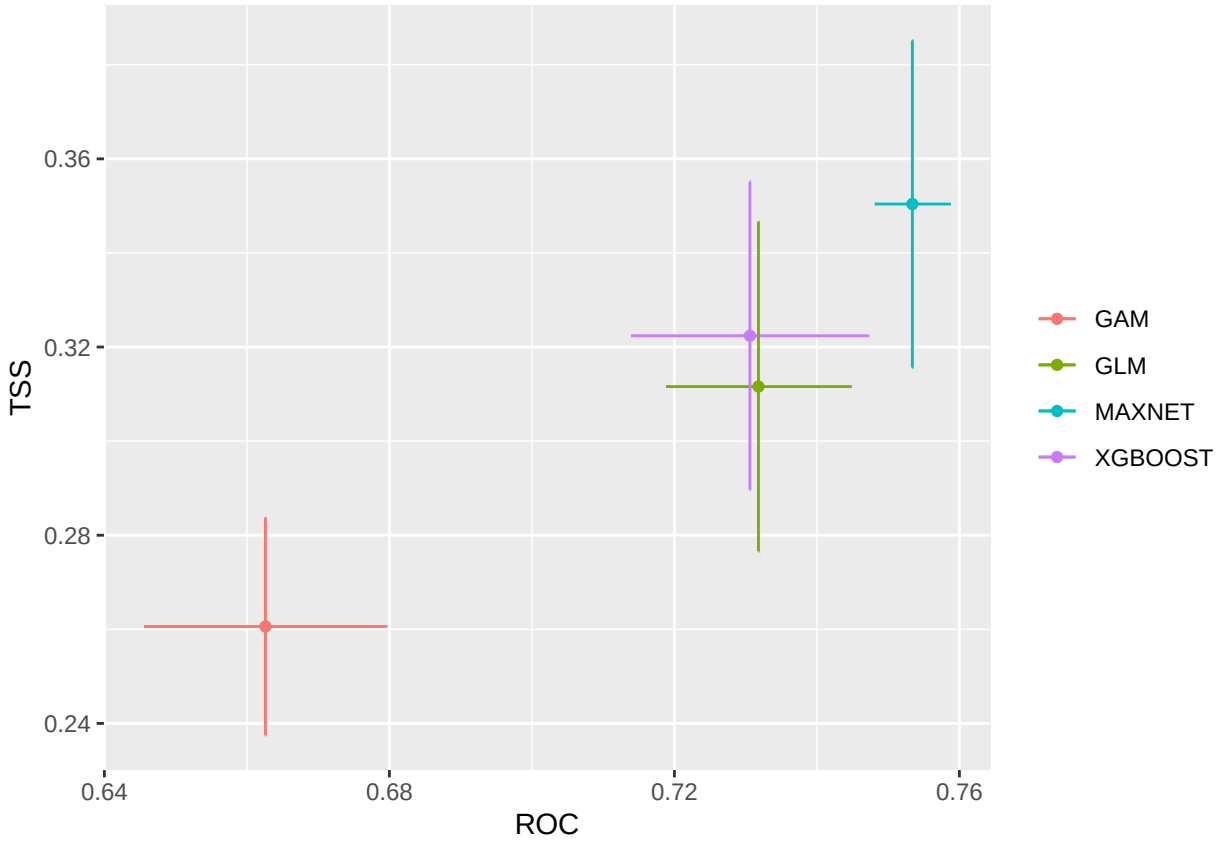
Vipera latastei

The code is basically the same, just changing the input data and directory names for *Vipera latastei*.

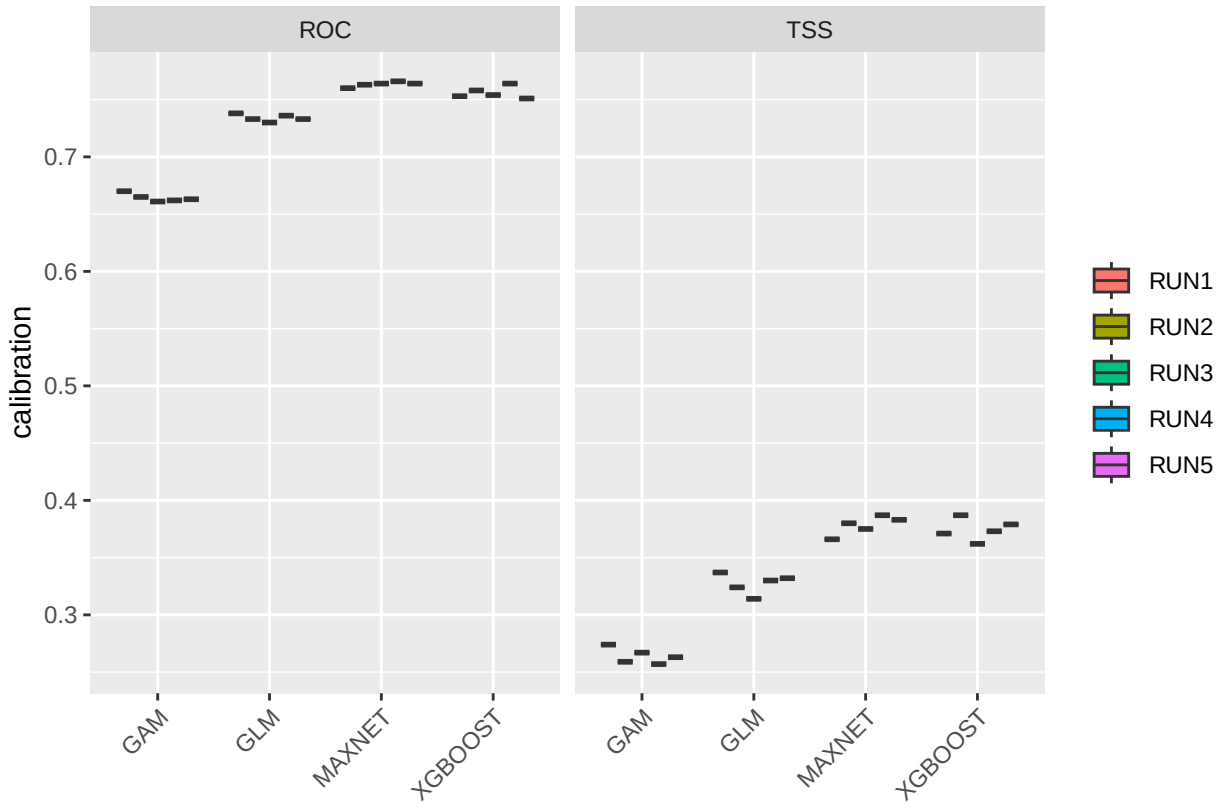
```r
n_vl <- sum(pres$species == "Vlatastei")
coords_vl <- pres[pres$species == "Vlatastei", 2:3]

vlData <- BIOMOD_FormatingData(resp.var = rep(1, n_vl),
                               expl.var = vars,
                               resp.xy = coords_vl,
                               resp.name = "Vlatastei",
                               dir.name = "models",
                               PA.nb.rep = 3,
                               PA.nb.absences = 10000,
                               PA.strategy = "disk",
                               PA.dist.max = 110000)

vlModel <- BIOMOD_Modeling(bm.format = vlData,
                           modeling.id = "EcoMod",
                           models = c("GAM", "GLM", "MAXNET", "XGBOOST"),
                           CV.strategy = "kfold",
                           CV.k = 5,
                           CV.do.full.models = FALSE,
                           OPT.strategy = "bigboss",
                           var.import = 3,
                           metric.eval = c("TSS", "ROC"))
# Plotting examples
plt <- bm_PlotEvalMean(vlModel, dataset="calibration")
```



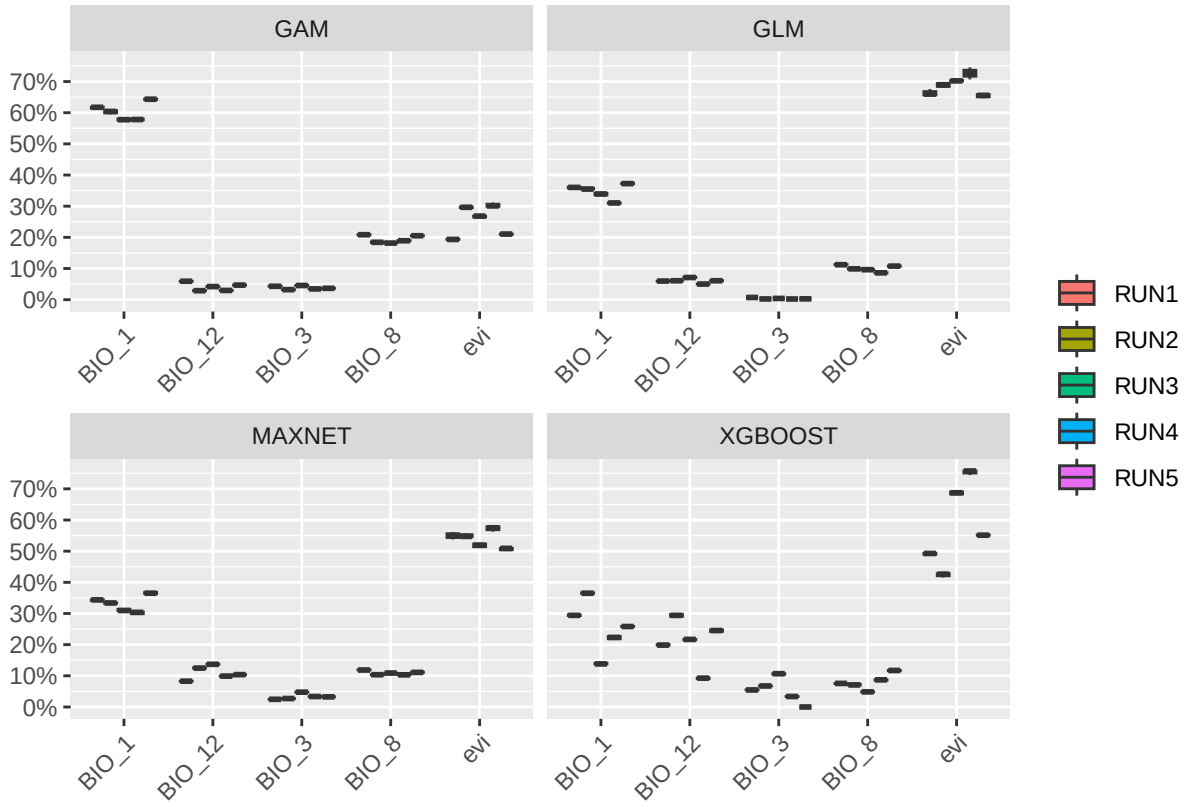
```
plt <- bm_PlotEvalMean(vlModel, dataset="validation")
```



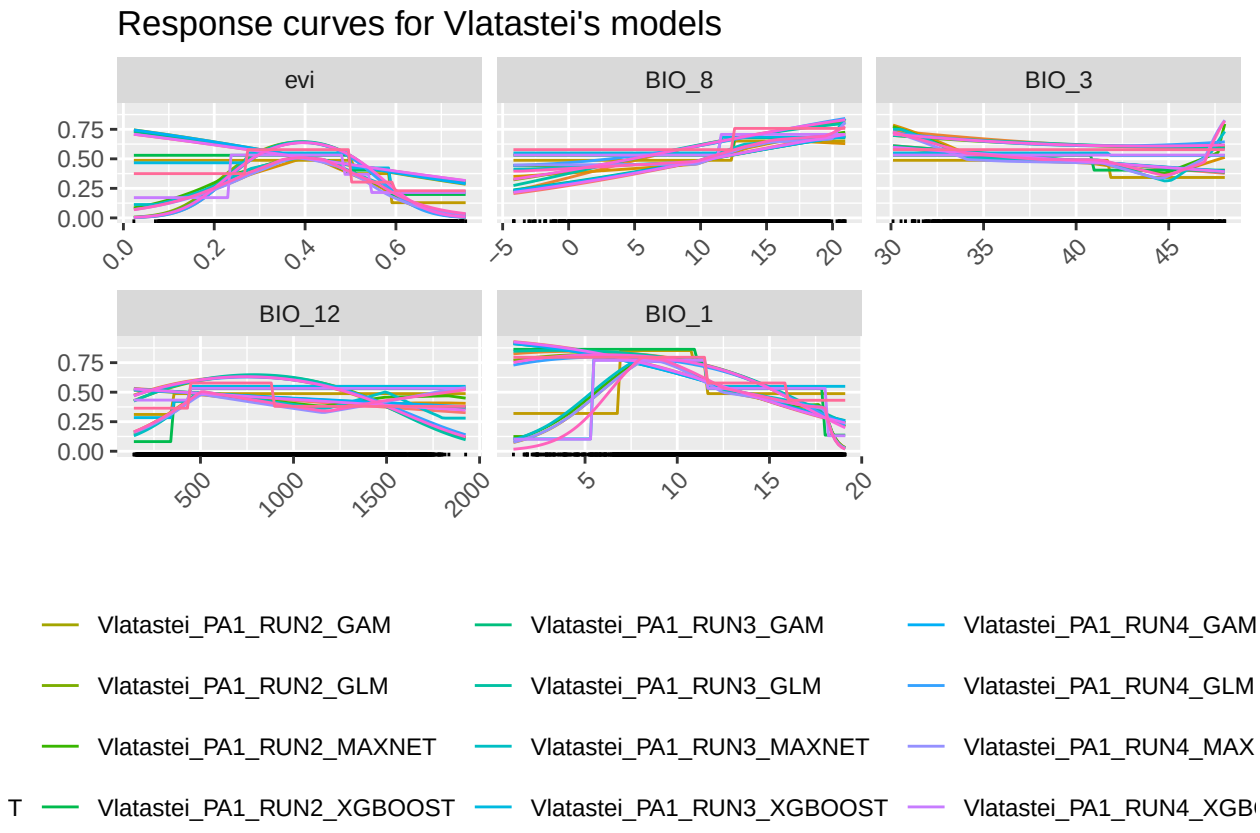

```
plt <- bm_PlotEvalBoxplot(vlModel, group.by = c('algo', 'run'))
```




```
plt <- bm_PlotVarImpBoxplot(vlModel, group.by = c('expl.var', 'run', 'algo'))
```



```
plt <- bm_PlotResponseCurves(vlModel, fixed.var = 'median')
```



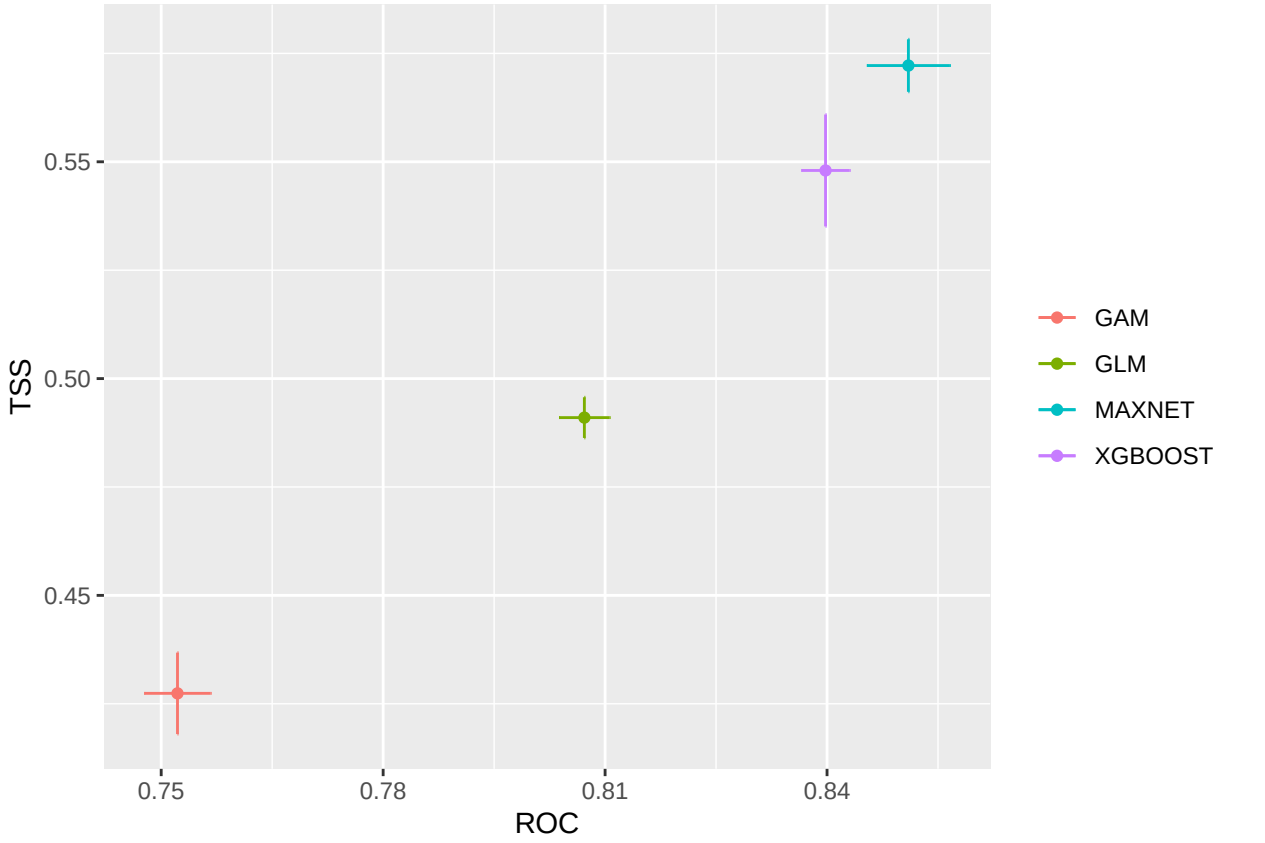

```
vlProj <- BIOMOD_Projection(bm.mod = vlModel,
                            proj.name = 'Present',
                            new.env = vars,
                            models.chosen = 'all')
```

Vipera seoanei

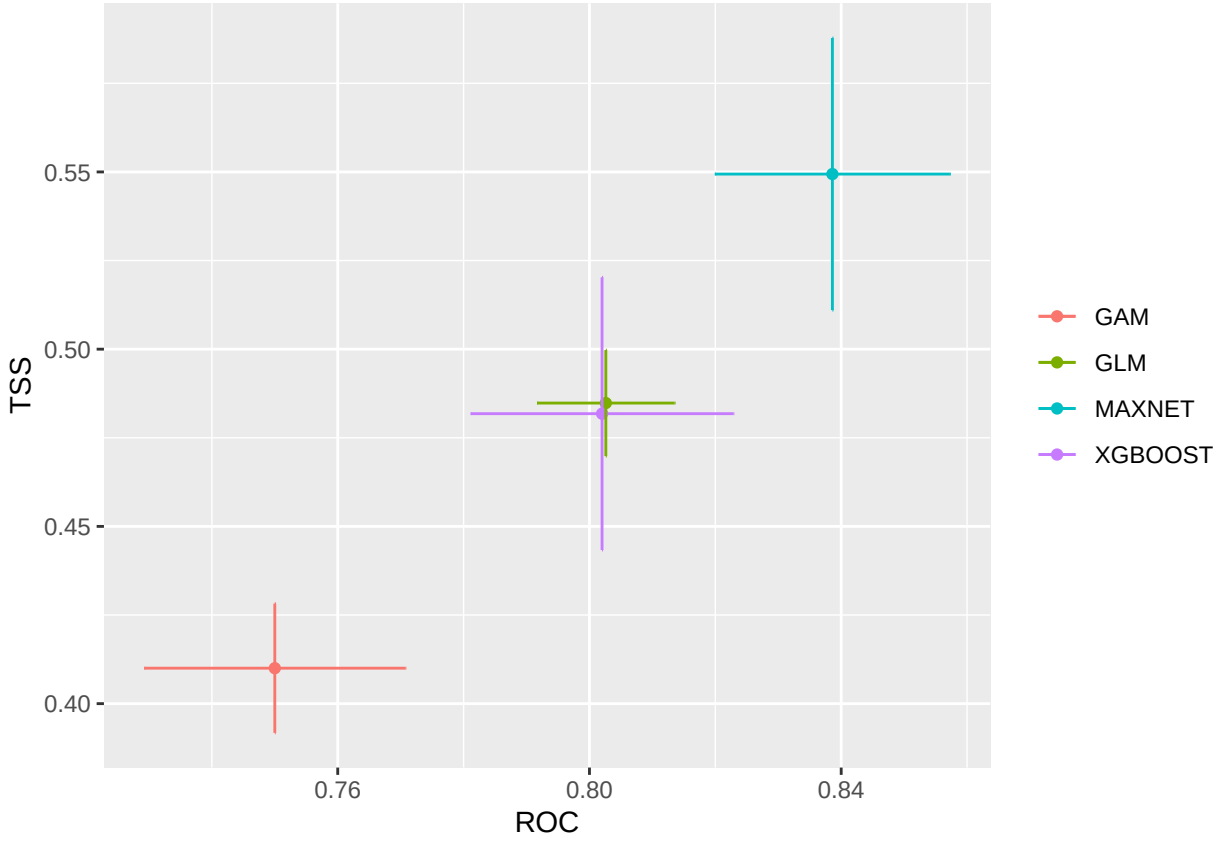
The input data is now for *Vipera seoanei*.

```
n_vs <- sum(pres$species == "Vseoanei")
coords_vs <- pres[pres$species == "Vseoanei", 2:3]

vsData <- BIOMOD_FormatingData(resp.var = rep(1, n_vs),
                               expl.var = vars,
                               resp.xy = coords_vs,
                               resp.name = "Vseoanei",
                               dir.name = "models",
                               PA.nb.rep = 3,
                               PA.nb.absences = 10000,
                               PA.strategy = "disk",
                               PA.dist.max = 110000)

vsModel <- BIOMOD_Modeling(bm.format = vsData,
                           modeling.id = "EcoMod",
                           models = c("GAM", "GLM", "MAXNET", "XGBOOST"),
                           CV.strategy = "kfold",
```

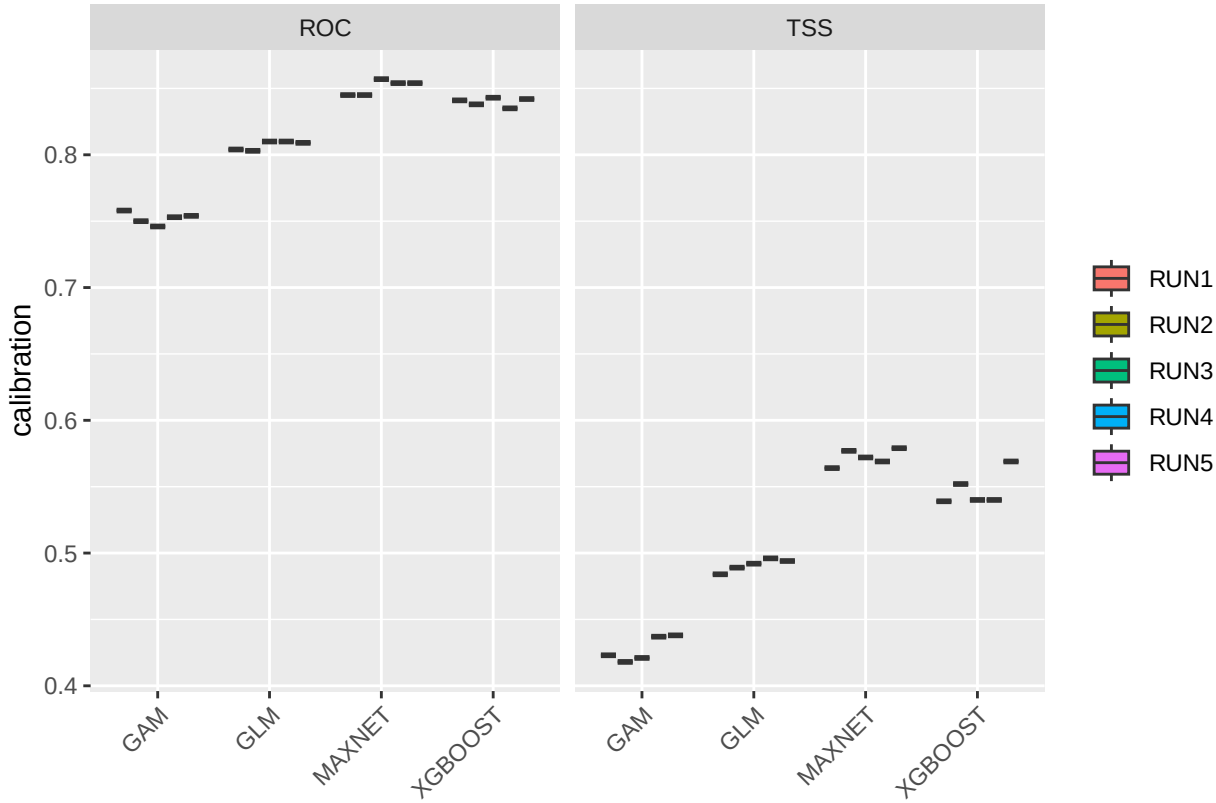
```
                          CV.k = 5,
                          CV.do.full.models = FALSE,
                          OPT.strategy = "bigboss",
                          var.import = 3,
                          metric.eval = c("TSS", "ROC"))
# Plotting examples
plt <- bm_PlotEvalMean(vsModel, dataset="calibration")
```



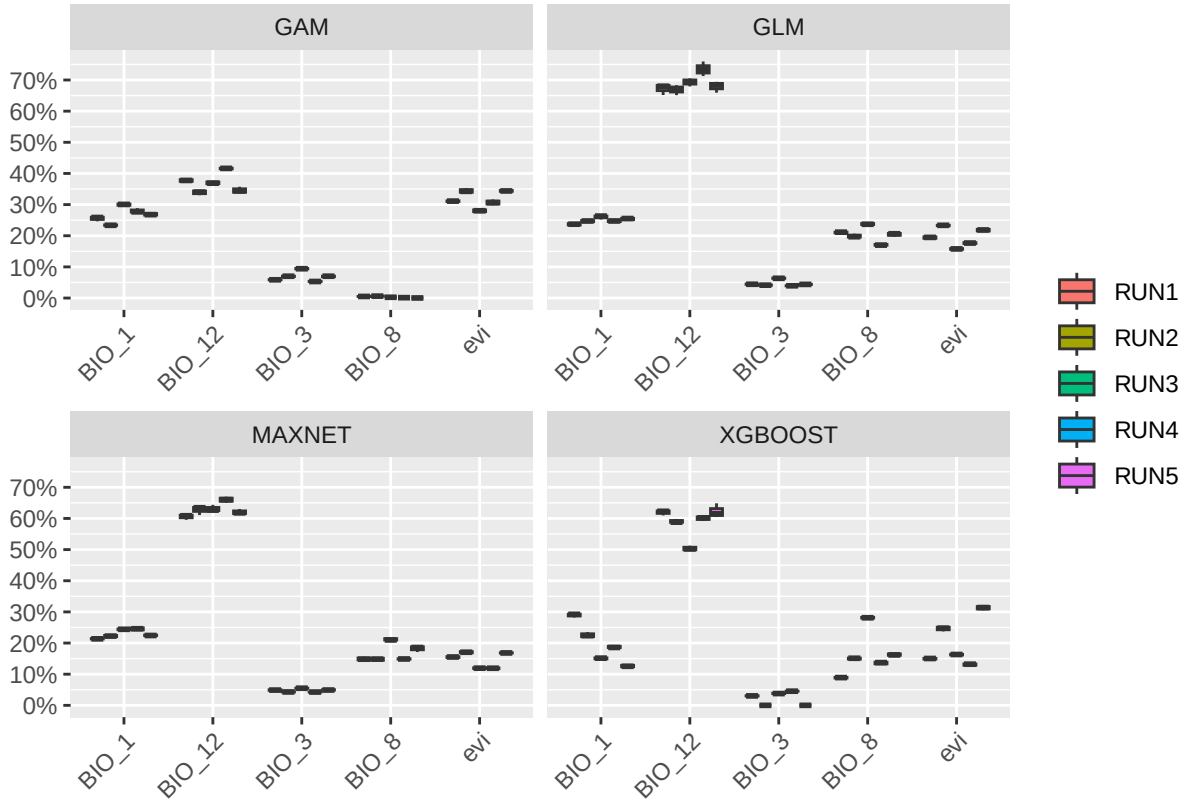

```
plt <- bm_PlotEvalMean(vsModel, dataset="validation")
```



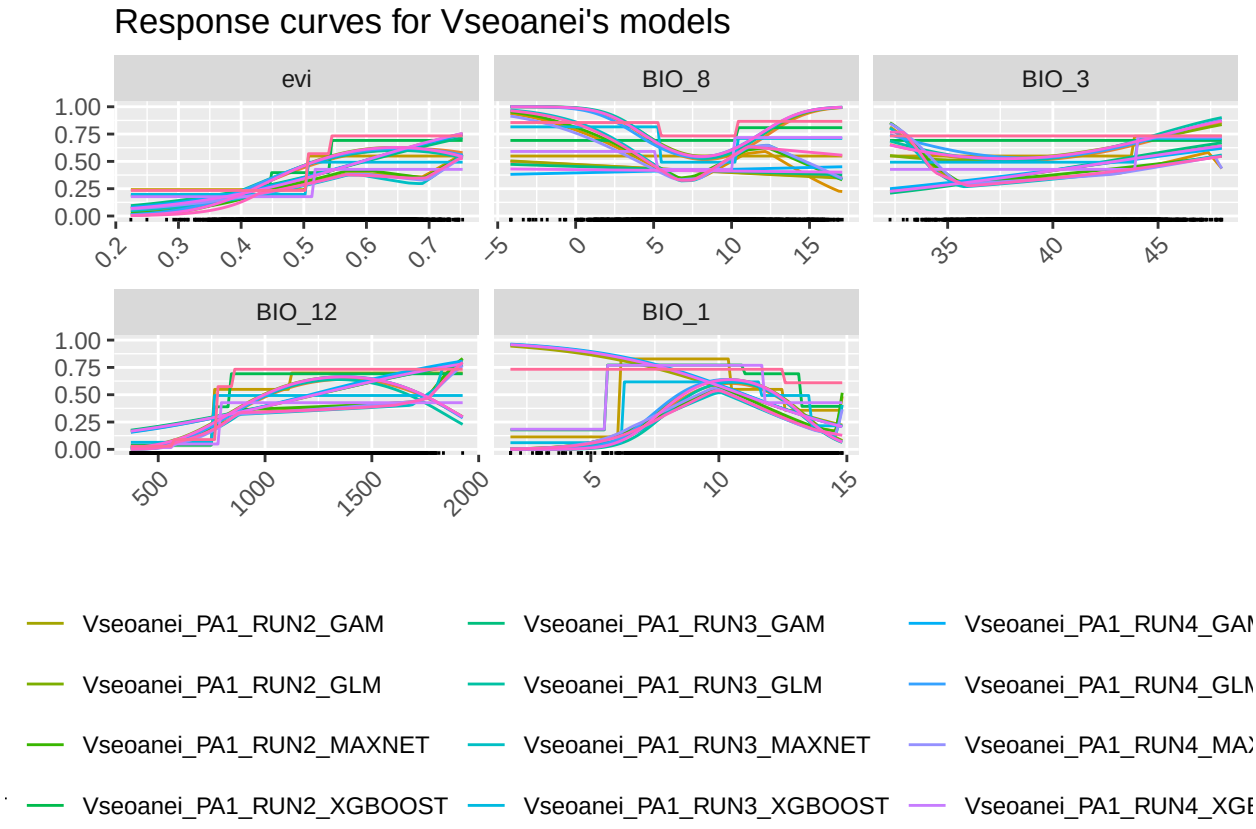

```
plt <- bm_PlotEvalBoxplot(vsModel, group.by = c('algo', 'run'))
```



```
plt <- bm_PlotVarImpBoxplot(vsModel, group.by = c('expl.var', 'run', 'algo'))
```




```
plt <- bm_PlotResponseCurves(vsModel, fixed.var = 'median')
```




```
vlProj <- BIOMOD_Projection(bm.mod = vsModel,
                            proj.name = 'Present',
                            new.env = vars,
                            models.chosen = 'all')
```

Final considerations for the modelling

Now, all models are available in the models directory. We will access them in the next chapters for ensembling, projecting, and plotting.

By examining the different plots for each species, one can observe that different variables have varying importance for predicting each species. This is a significant finding, as it allows us to describe the niche of each species in terms of environmental preferences. Some species' models will be more dependent on climate variables, making their distributions more susceptible to climate change. Species that rely more on static variables (NDVI, in this example) will have distributions in projections for other time periods that are less prone to change.

Chapter 7 - Model ensemble

In this chapter, we require the 60 models resulting from the combinations of pseudo-absence sets, modeling algorithms, and resampling strategies for each species. Ensembling models involves summarizing all this information into a single prediction. This allows us to capture a wide range of possible responses arising from the different combinations. Some algorithms have a stochastic nature, like machine learning methods, and even without changing any other conditions, running the model with the same data might generate slightly different predictions. Other algorithms are more static, such as GLM: running the model with exactly the same data will generate the same prediction. That's one reason we vary the initial conditions of the model (e.g., pseudo-absences dataset and resampling). This wider range of predictions is supposed to capture slight variations in information about the species' niche and thus create a more robust final model.

For this chapter we will need again the *biomod2* package.

```
library(biomod2)
```

Ensembling *Vipera aspis*

Assuming that we have a clean workspace, we have to load the models from the folder where they were saved *models/vaspis*. As *biomod2* deals with very different algorithms and data, it saves many information and data that might be cumbersome to understand what is in the folder. However, by importing into R one of the files, we can access all information through *biomod2* functions. It requires a small coding trick to fully import the model data with a common object name. We have to load and retrive that data in two lines of code:

```
vaName <- load("models/Vaspis/Vaspis.EcoMod.models.out")
vaModel <- eval(str2lang(vaName))
vaModel
```

The object `vaModel` contains all the models built in the previous chapter. Now, we can proceed with ensembling. The package provides a straightforward function with customizable arguments for ensembling. We specify the models we want to ensemble and choose the type of ensembling. In this example, we are ensembling all models together, meaning that all combinations of pseudo-absences, algorithms, and runs are merged. However, we could choose to ensemble only specific subsets, such as by algorithm, by setting `em.by = "PA_dataset+repet"`, which would result in separate ensembles for each algorithm. This approach could be useful for evaluating the performance of each algorithm individually. Here, we are ensembling all models together using `em.by = "all"` while `models.chosen = 'all'` aensures that all built models are available for ensembling.

We select the median as the ensembling statistic. Other options available include mean and weighted mean, where weights are typically based on model performance metrics or confidence intervals.

Since the resulting ensemble model provides predictions that have not yet been tested, we evaluate its performance using the same metrics as before (TSS and ROC). Variable importance is also assessed using 3 permutations, as in previous steps.

```
vaEnsbl <- BIOMOD_EnsembleModeling(bm.mod = vaModel,
                                   models.chosen = 'all',
                                   em.by = 'all',
                                   em.algo = 'EMmedian',
                                   metric.eval =  c('TSS', 'ROC'),
                                   var.import = 3)
```

```
vaEnsbl
```

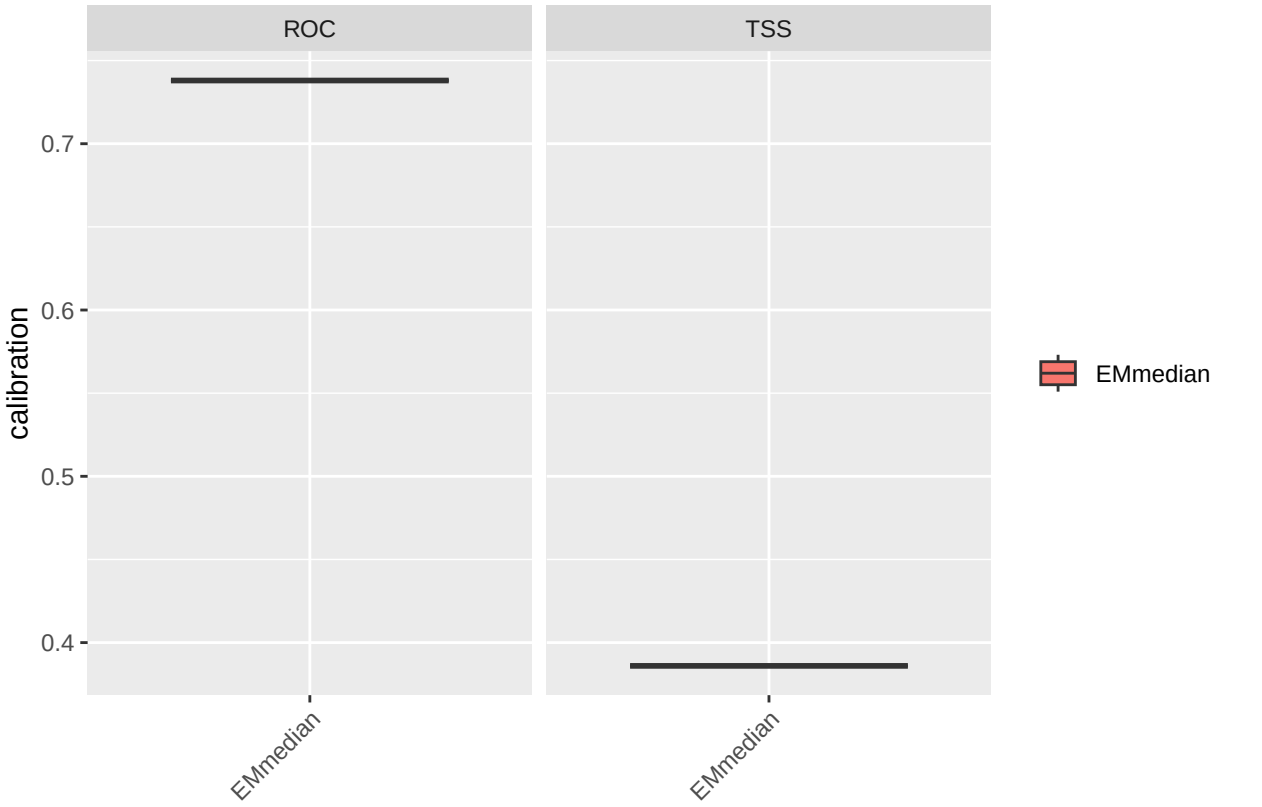
```
##
## -=-=-=-=-=-=-=-=-=-=-=-=-=-= BIOMOD.ensemble.models.out -=-=-=-=-=-=-=-=-=-=-=-=-=-=
##
## sp.name : Vaspis
##
## expl.var.names : evi BIO_8 BIO_3 BIO_12 BIO_1
##
##
## models computed:
## Vaspis_EMmedianByTSS_mergedData_mergedRun_mergedAlgo, Vaspis_EMmedianByROC_mergedData_mergedRun_merge
##
## models failed: none
##
## -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

At this stage, we should have a new file in the *models/Vaspis* folder containing the ensembling results, which

we can access later as needed.

We can evaluate the ensembled model by plotting its performance metrics. Now we don't have calibration and validation datasets and we test with full data.

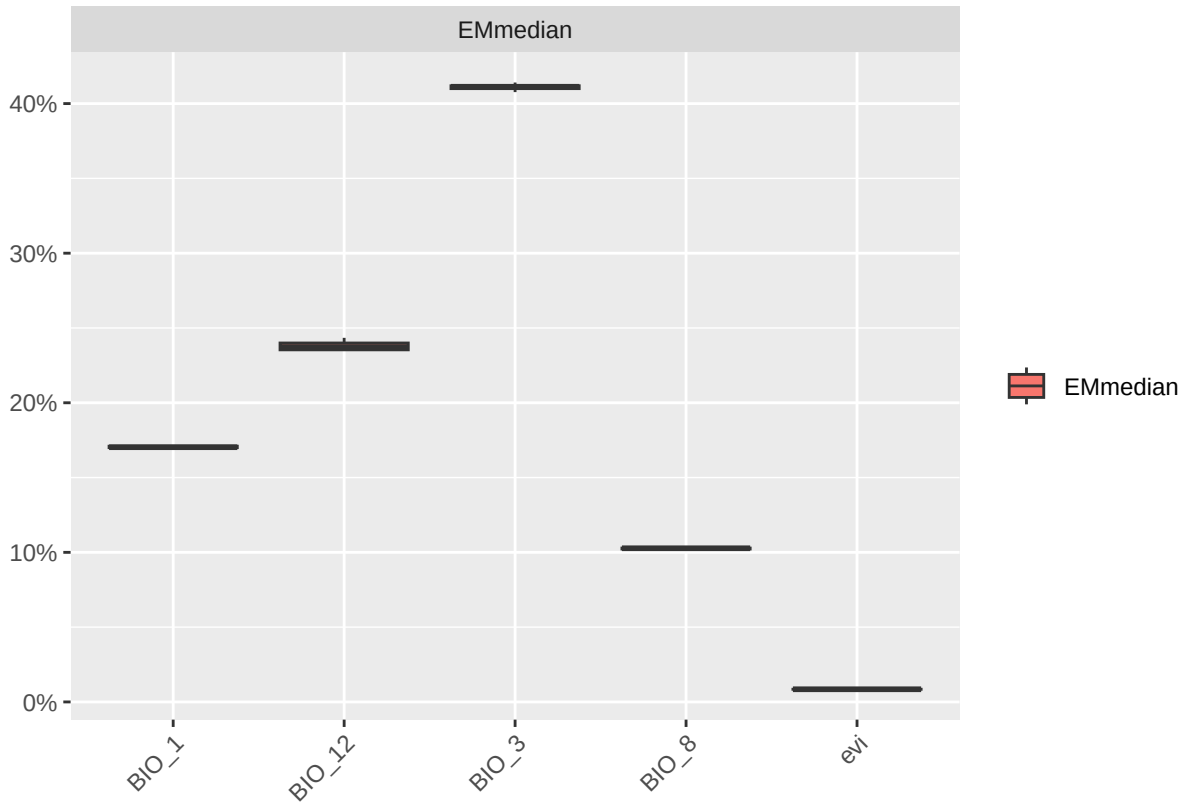
```
plt <- bm_PlotEvalBoxplot(vaEnsbl, group.by = c('algo', 'algo'))
```



Both metrics show reasonable performance. However, at this stage, we should consider adding more predictors or potentially removing some to improve model performance. If an important predictor defining the species niche is missing, it could significantly affect performance. Another strategy could involve adjusting the width of the buffer for pseudo-absences. Increasing environmental variability might facilitate model fitting, but this approach should be carefully balanced and justified (as it might be a contentious subject!).

We can also check the variable importance.

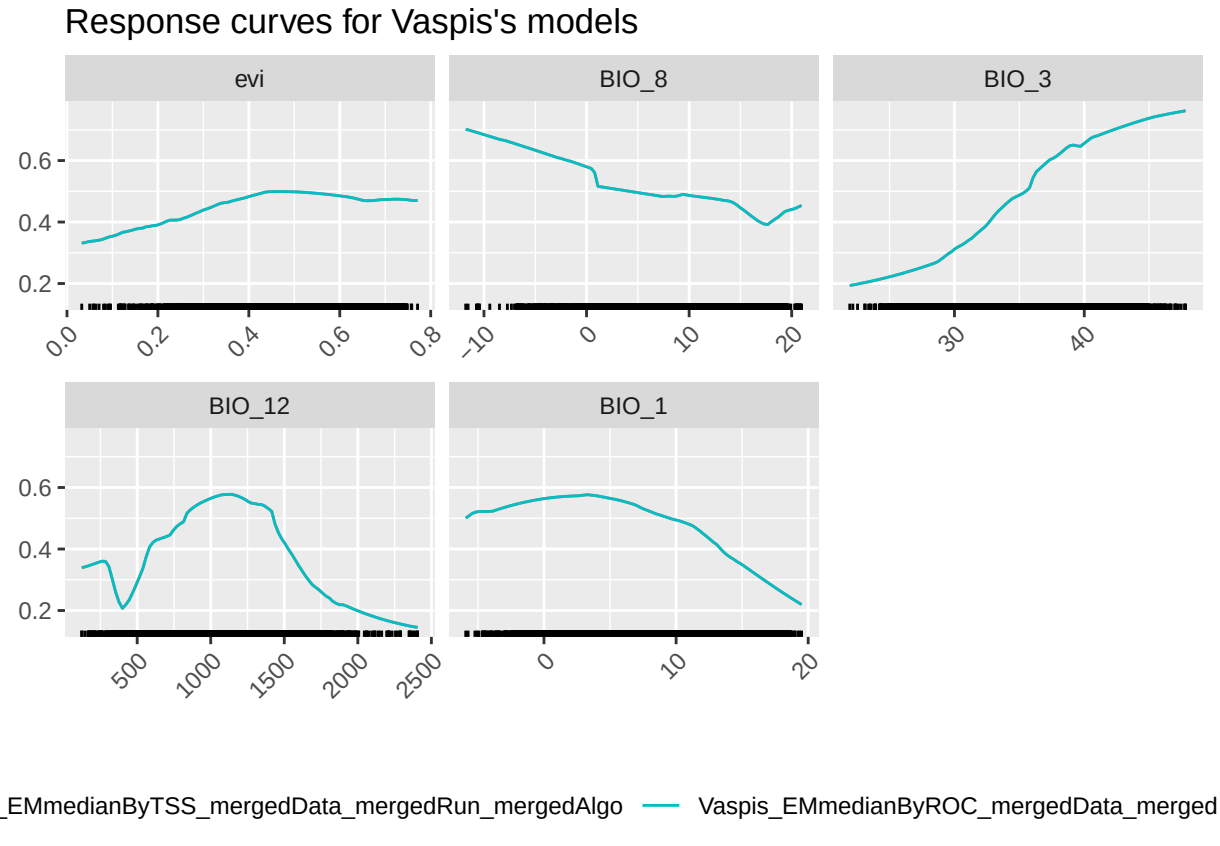
```
plt <- bm_PlotVarImpBoxplot(vaEnsbl, group.by = c('expl.var', 'algo', 'algo'))
```



In this model, BIO_12 and BIO_7 are clearly the most important variables, while NDVI contributes the least to the models."

And finally we can check variable response curves.

```
plt <- bm_PlotResponseCurves(vaEnsbl, fixed.var = 'median')
```

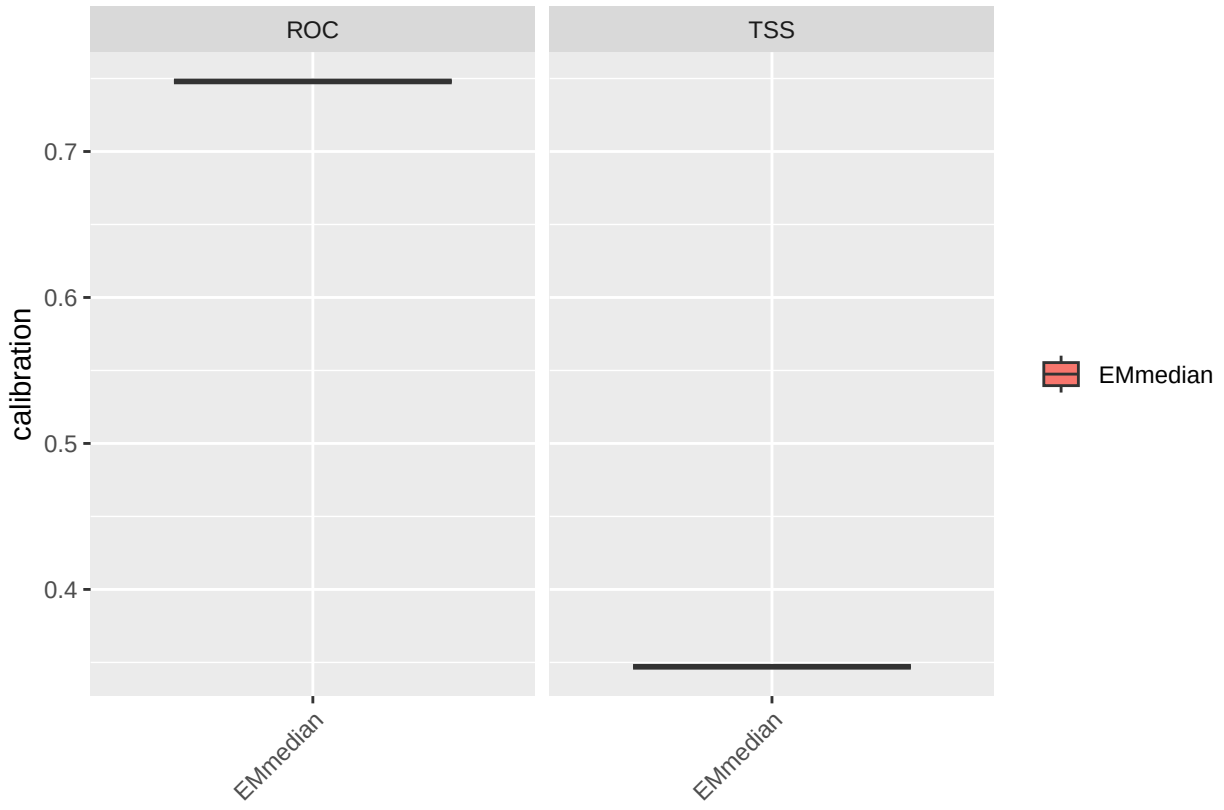


The ensembled models provide a single response curve for each variable, which is easier to interpret than before. We should give more importance to the BIO_12 and BIO_7 curves, as those are the variables that contribute more to the models. The model seems to increase the predictive probability with the increase of the value of these variables. Notice how the NDVI, which is the least important variable, appears to have a trend, but the prediction range associated with it is low.

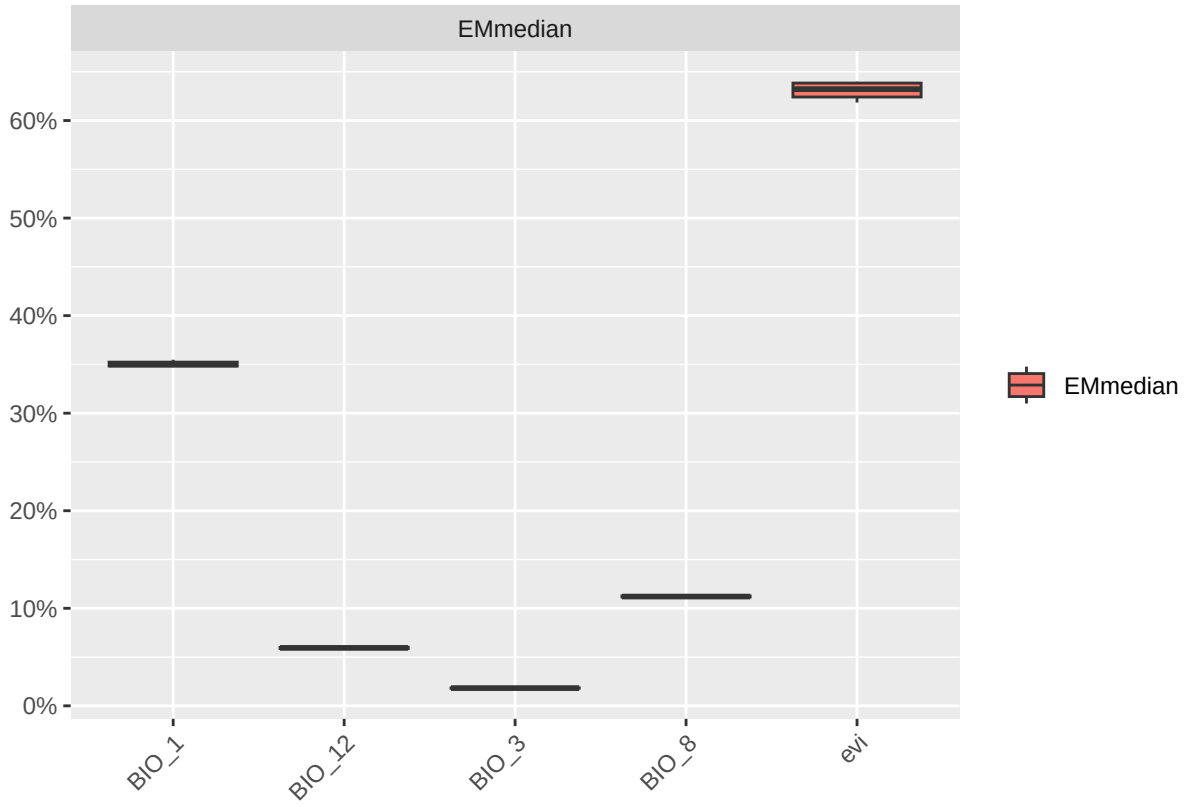
Ensembling *Vipera latastei*

Repeat the code, changing the model.

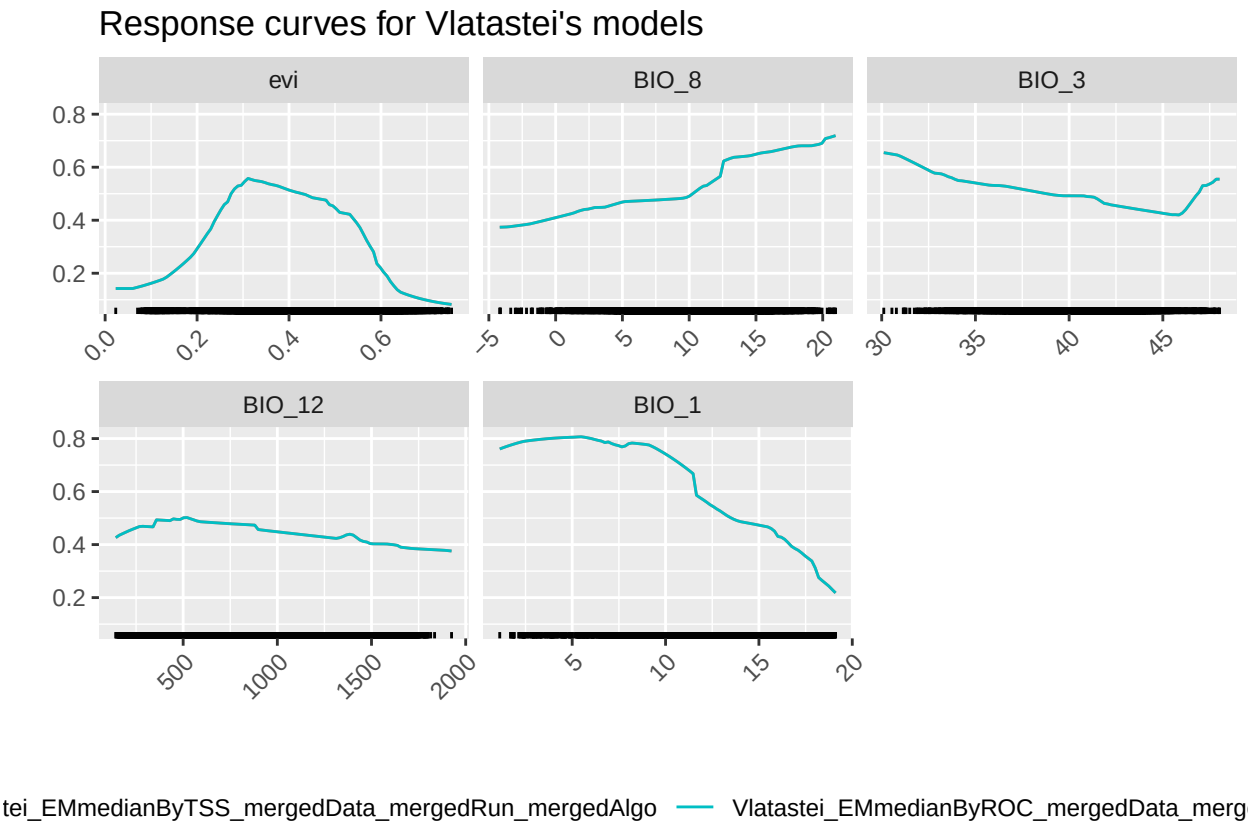
```
vlName <- load("models/Vlatastei/Vlatastei.EcoMod.models.out")
vlModel <- eval(str2lang(vlName))


# Ensemble
vlEnsbl <- BIOMOD_EnsembleModeling(bm.mod = vlModel,
                                   models.chosen = 'all',
                                   em.by = 'all',
                                   em.algo = 'EMmedian',
                                   metric.eval = c('TSS', 'ROC'),
                                   var.import = 3)


# Plotting examples
plt <- bm_PlotEvalBoxplot(vlEnsbl, group.by = c('algo', 'algo'))
```




```
plt <- bm_PlotVarImpBoxplot(vlEnsbl, group.by = c('expl.var', 'algo', 'algo'))
```



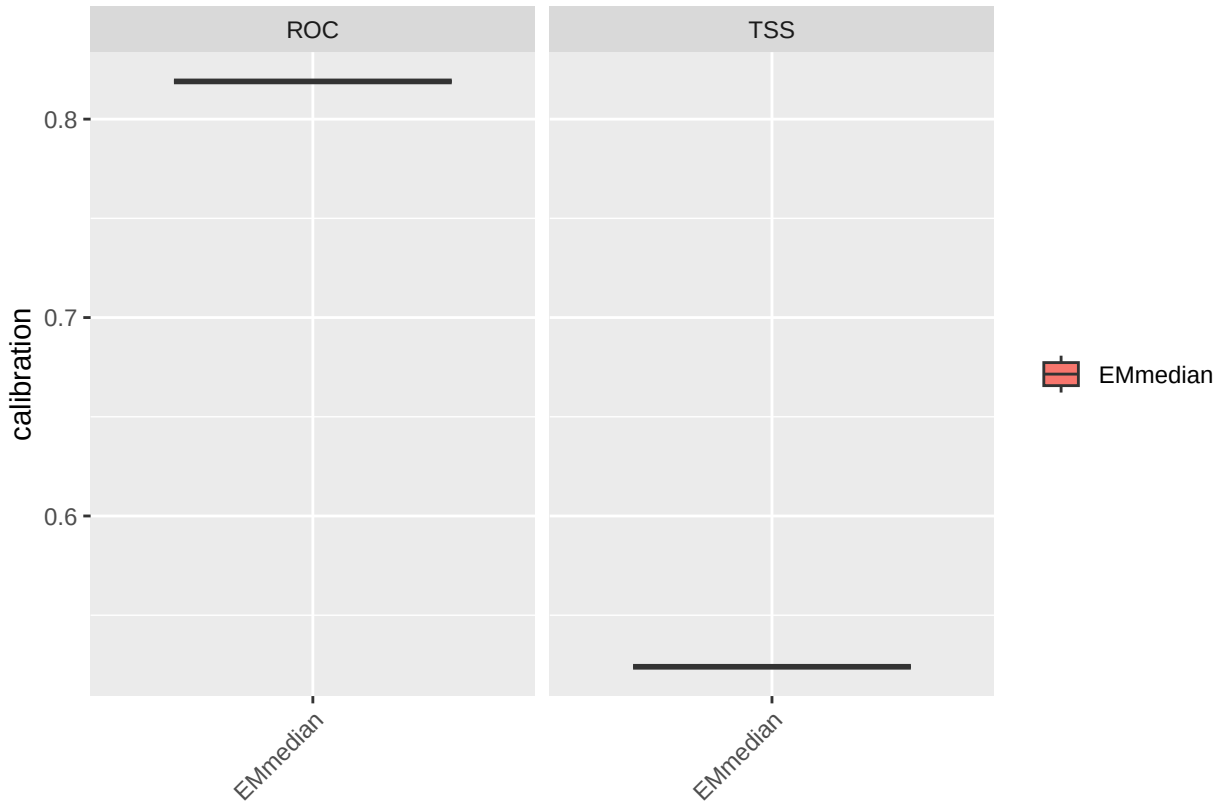
```
plt <- bm_PlotResponseCurves(vlEnsbl, fixed.var = 'median')
```



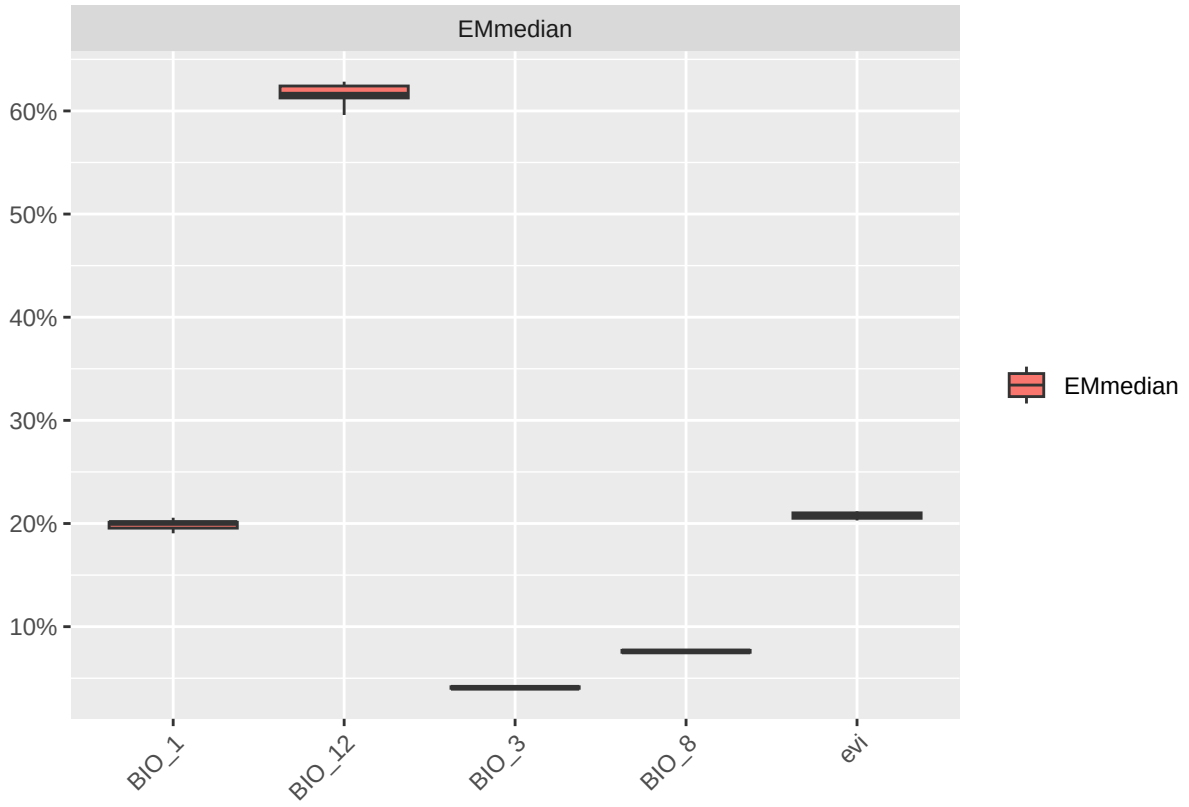
Ensembling *Vipera seoanei*

And for the third species:

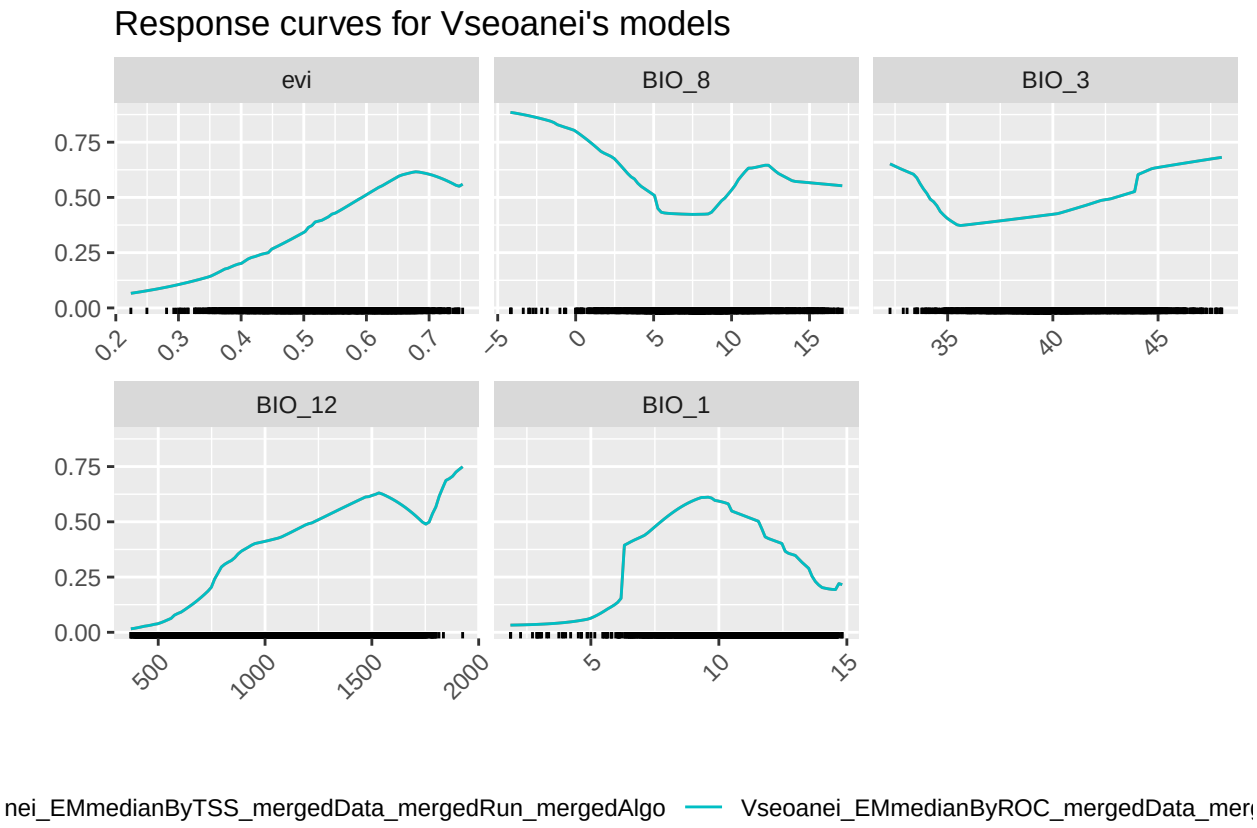
```
vsName <- load("models/Vseoanei/Vseoanei.EcoMod.models.out")
vsModel <- eval(str2lang(vsName))


# Ensemble
vsEnsbl <- BIOMOD_EnsembleModeling(bm.mod = vsModel,
                                   models.chosen = 'all',
                                   em.by = 'all',
                                   em.algo = 'EMmedian',
                                   metric.eval =  c('TSS', 'ROC'),
                                   var.import = 3)


# Plotting examples
plt <- bm_PlotEvalBoxplot(vsEnsbl, group.by = c('algo', 'algo'))
```




```
plt <- bm_PlotVarImpBoxplot(vsEnsbl, group.by = c('expl.var', 'algo', 'algo'))
```



```
plt <- bm_PlotResponseCurves(vsEnsbl, fixed.var = 'median')
```



Final considerations for the ensembling

The ensembling process reduces the complexity of having multiple models to analyze while ensuring the robustness of predictions. With the model ensembles now constructed, we are ready to project these models onto the Iberian Peninsula.

Chapter 8 - Model projections

In this chapter, we are going to use the ensemble models to project onto our region of interest, which is the Iberian Peninsula. Additionally, we have multiple ages and SSPs to project. By the end of this chapter, we will have projections for

- Current conditions
- Period 2011-2040
  - SSP 126
  - SSP 585
- Period 2041-2060
  - SSP 126
  - SSP 485
- Period 2061-2080
  - SSP 126
  - SSP 586

This results in 7 model projections for the Iberian Peninsula.

We need both *terra* library for opening the varaibles, and *biomod2* for interpreting the models.

```
library(terra)
library(biomod2)
```

We can define here some variables that will be useful for looping over ages and SSPs (similar to Chapter 5) to read and project models to different time periods."

```
ages <- c("2021-2040", "2041-2060", "2061-2080")
ssps <- c("126", "585")
```

Again, we will proceed with the first species and then run the same code for the other two to obtain all projections.

We can open the raster for the current period variables as this will be constant for all current projections.

```
curp <- rast("data/rasters/proj_current.tif")
```

Projecting *Vipera aspis* models

WE use the same trick as in the previous chapter to open the ensembled model available in the *models/Vaspis* folder:

```
vaName <- load("models/Vaspis/Vaspis.EcoMod.ensemble.models.out")
vaEnsbl <- eval(str2lang(vaName))
```

Until now, we have been focusing in the continuous probability predictions. However, models can be converted to binary (presence vs. absence) by setting a threshold that results in best performance. We can check which threshold *biomod2* is using for these models.

```
get_evaluations(vaEnsbl)
```

```
##                                              full.name merged.by.PA
## 1 Vaspis_EMmedianByTSS_mergedData_mergedRun_mergedAlgo   mergedData
## 2 Vaspis_EMmedianByTSS_mergedData_mergedRun_mergedAlgo   mergedData
## 3 Vaspis_EMmedianByROC_mergedData_mergedRun_mergedAlgo   mergedData
## 4 Vaspis_EMmedianByROC_mergedData_mergedRun_mergedAlgo   mergedData
##   merged.by.run merged.by.algo filtered.by     algo metric.eval cutoff
## 1     mergedRun     mergedAlgo         TSS EMmedian         TSS  491.0
## 2     mergedRun     mergedAlgo         TSS EMmedian         ROC  509.5
## 3     mergedRun     mergedAlgo         ROC EMmedian         TSS  491.0
## 4     mergedRun     mergedAlgo         ROC EMmedian         ROC  509.5
##   sensitivity specificity calibration validation evaluation
## 1      74.019      64.541       0.386         NA         NA
## 2      69.672      68.945       0.738         NA         NA
## 3      74.019      64.541       0.386         NA         NA
## 4      69.672      68.945       0.738         NA         NA
```

Depending on the metric, the threshold varies as it has to maximize the performance of each metric separately. For TSS, the threshold (cutoff) is set to 0.474, and for ROC, it is set to 0.468. Note that *biomod2* multiplies the probabilities by 1000, so they range between 0 and 1000.

We can project the model to current conditions in the Iberian Peninsula. The function will create both continuous predictions and binary predictions using the thresholds mentioned above. It is important to name the projections appropriately for easy identification of the rasters. We set the `proj.name` to **Current** and a new folder named *proj_Current* will be created inside *models/Vaspis*. In this folder, two rasters will be saved for continuous and binary predictions. Also, the argument *new.env* refers to the new environmental conditions that are the raster variables we want to project to.

```
vaProj <- BIOMOD_EnsembleForecasting(bm.em = vaEnsbl,
                                     proj.name = 'Current',
                                     new.env = curp,
                                     models.chosen = 'all',
                                     metric.binary = 'TSS')
```

We will have to do the same for each combination of age and SSP projections. For that, we use a nested for loop as before (chapter 5). The logic is as follows:

- For each combination of age and ssp
    1. Construct the respective filename using age and SSP and open the raster.
    2. Project the ensembled model to the respective time period, correctly naming the projection with age_ssp.

```
for (age in ages) {
    for (ssp in ssps) {
        projVars <- rast(paste0("data/rasters/proj_", age, "_", ssp, ".tif"))
        vaProj <- BIOMOD_EnsembleForecasting(bm.em = vaEnsbl,
                                             proj.name = paste0(age, "_", ssp),
                                             new.env = projVars,
                                             models.chosen = 'all',
                                             metric.binary = 'TSS')
    }
}
```

At this stage, we should have a folder inside *models/Vaspis* for each projection that we built. We now need to run the code for the other two species to complete all projections.

Projecting *Vipera latastei* models

We load the *Vipera latastei* ensembled model and run the code.

```
vlName <- load("models/Vlatastei/Vlatastei.EcoMod.ensemble.models.out")
vlEnsbl <- eval(str2lang(vlName))

get_evaluations(vlEnsbl)

# Project to current time
vlProj <- BIOMOD_EnsembleForecasting(bm.em = vlEnsbl,
                                     proj.name = 'Current',
                                     new.env = curp,
                                     models.chosen = 'all',
                                     metric.binary = 'TSS')
# Project to future periods
for (age in ages) {
    for (ssp in ssps) {
        projVars <- rast(paste0("data/rasters/proj_", age, "_", ssp, ".tif"))
        vlProj <- BIOMOD_EnsembleForecasting(bm.em = vlEnsbl,
                                             proj.name = paste0(age, "_", ssp),
                                             new.env = projVars,
                                             models.chosen = 'all',
                                             metric.binary = 'TSS')
    }
}
```

Projecting *Vipera latastei* models

Nowe for *Vipera seoanei*.

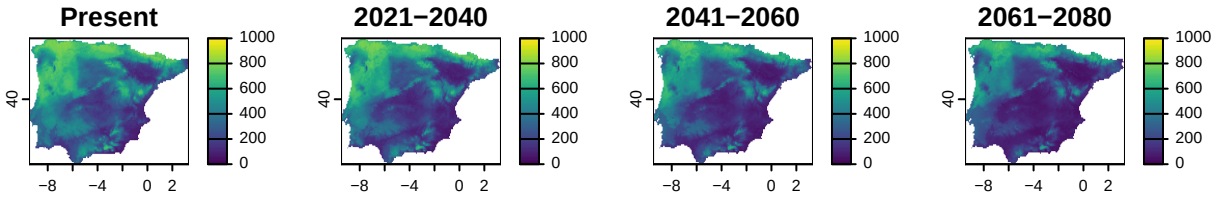
```
vsName <- load("models/Vseoanei/Vseoanei.EcoMod.ensemble.models.out")
vsEnsbl <- eval(str2lang(vsName))

get_evaluations(vsEnsbl)

# Project to current time
vsProj <- BIOMOD_EnsembleForecasting(bm.em = vsEnsbl,
                                     proj.name = 'Current',
                                     new.env = curp,
                                     models.chosen = 'all',
                                     metric.binary = 'TSS')
# Project for future
for (age in ages) {
    for (ssp in ssps) {
        projVars <- rast(paste0("data/rasters/proj_", age, "_", ssp, ".tif"))
        vsProj <- BIOMOD_EnsembleForecasting(bm.em = vsEnsbl,
                                             proj.name = paste0(age, "_", ssp),
                                             new.env = projVars,
                                             models.chosen = 'all',
                                             metric.binary = 'TSS')
    }
}
```

Final considerations for projections

At this point, we have all the projections needed to answer our initial question. By defining a smaller projection zone (the Iberian Peninsula) relevant to our research question, we have optimized both computer storage and processing time, as we avoid unnecessary projections over a larger area.

All files are saved in respective folders and we can access them using any GIS. As an example, here are the model projections for *Vipera aspis* for the current and future periods, only for scenario ssp 585.

```
p1 <- rast("models/Vaspis/proj_Current/proj_Current_Vaspis_ensemble.tif")
p2 <- rast("models/Vaspis/proj_2021-2040_585/proj_2021-2040_585_Vaspis_ensemble.tif")
p3 <- rast("models/Vaspis/proj_2041-2060_585/proj_2041-2060_585_Vaspis_ensemble.tif")
p4 <- rast("models/Vaspis/proj_2061-2080_585/proj_2061-2080_585_Vaspis_ensemble.tif")

layout(matrix(1:4, 1))
plot(p1[[1]], main="Present", range=c(0,1000))
plot(p2[[1]], main="2021-2040", range=c(0,1000))
plot(p3[[1]], main="2041-2060", range=c(0,1000))
plot(p4[[1]], main="2061-2080", range=c(0,1000))
```



Chapter 9 - Finally, answering the question

In this chapter, we will open all projection rasters created previously and analyze how these predictions can help identify sympatric areas for the three species. As the results showed, the models reveal different relationships with the predictors, resulting in distinct distributions. How do these potential distributions intersect with each other?

We will use the terra package since most operations in this chapter involve geoprocessing with rasters. While any GIS software could be used, using terra allows us to automate some processes.

```
library(terra)
```

We can also prepare some variables to loop over future periods, as we did before.

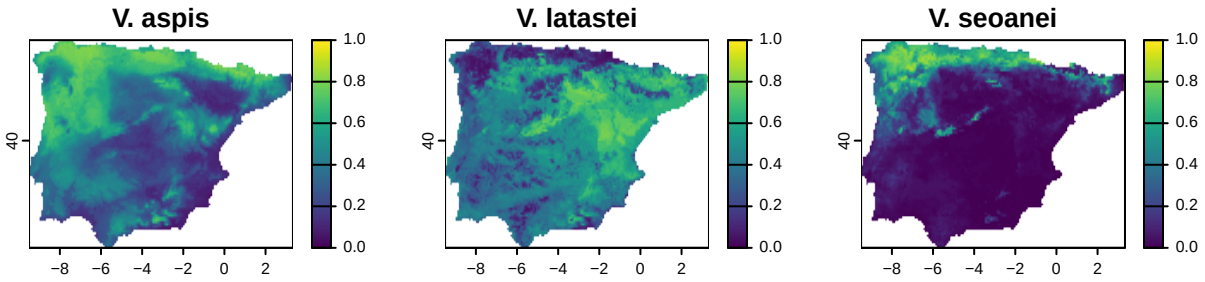
```
ages <- c("2021-2040", "2041-2060", "2061-2080")
ssps <- c("126", "585")
```

We can begin by importing the rasters of the current projections for the three species. We divide by 1000 to get the projections in the range of 0 to 1.

```
va <- rast("models/Vaspis/proj_Current/proj_Current_Vaspis_ensemble.tif")[[1]]
va <- va / 1000
vl <- rast("models/Vlatastei/proj_Current/proj_Current_Vlatastei_ensemble.tif")[[1]]
vl <- vl / 1000
vs <- rast("models/Vseoanei/proj_Current/proj_Current_Vseoanei_ensemble.tif")[[1]]
vs <- vs / 1000
```

We can plot the maps:

```
layout(matrix(1:3, 1))
plot(va, main="V. aspis", range=c(0,1))
plot(vl, main="V. latastei", range=c(0,1))
plot(vs, main="V. seoanei", range=c(0,1))
```

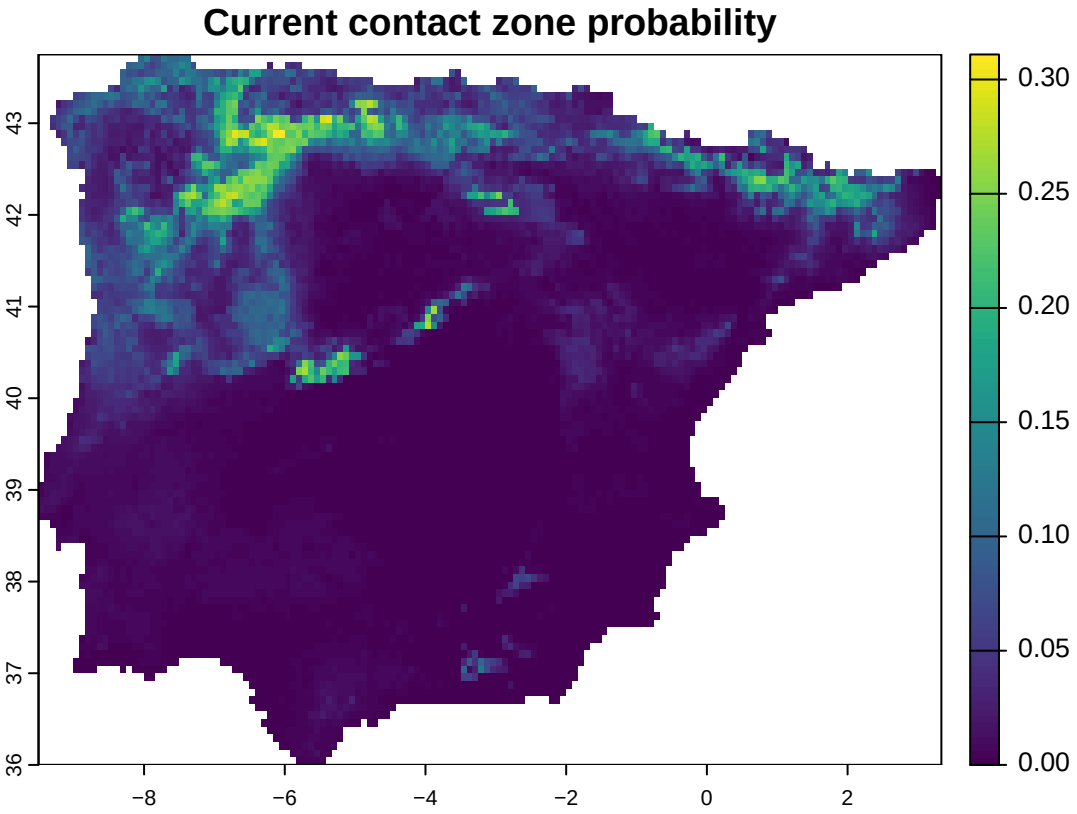


We need some processing to determine the sympatric zone between the vipers. Since we are using continuous probability maps, an easy way to estimate the maximum probability of finding all three species together is by calculating the product of the three maps. The potential maximum score would be 1, where all species have the highest probability, while any location with a probability of 0 for any species will have a sympatry probability of zero.

Additionally, there are other methods to calculate sympatry. For instance, using the binary presence/absence maps produced earlier, the sum of all maps would provide a range of values from 0 to 3, indicating how many species are present at each location.

Here, we will calculate the product of the probabilities:

```
CZ <- va*vl*vs
names(CZ) <- "Present"
plot(CZ, main= "Current contact zone probability")
```



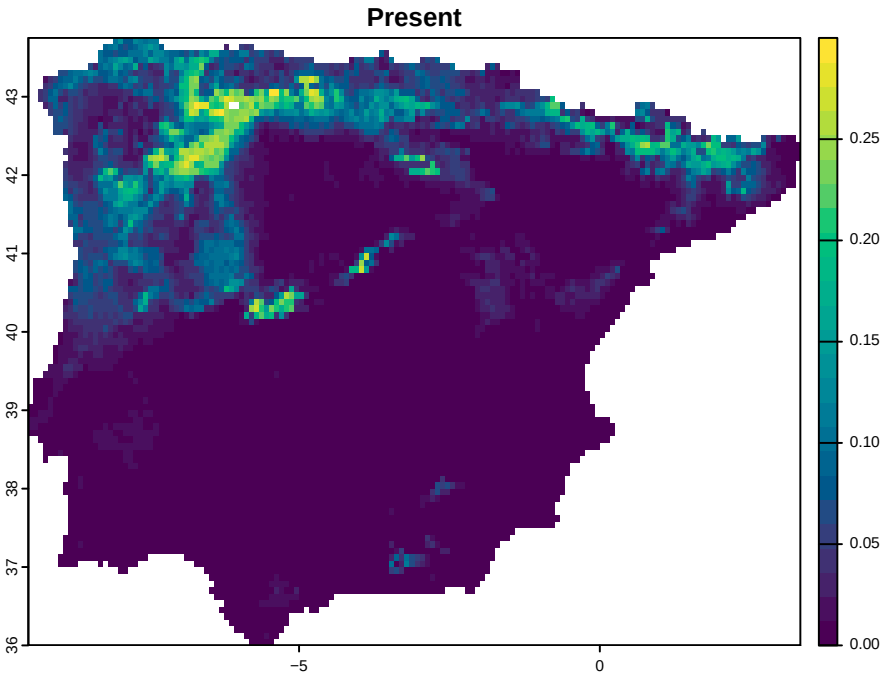
We have to calculate the same for all combinations of age and ssp projections. For that we use the same style of nested loop we have been using. We will be adding layers to CZ raster with correct names, following the age_ssp order.

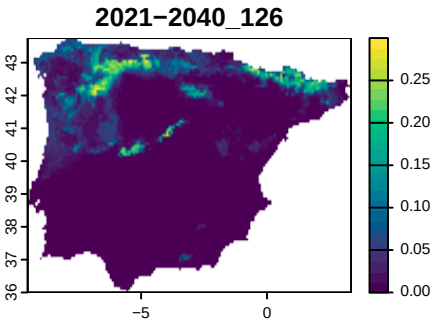
The logic of the loop is the following: * for each combination of age and ssp: 1. Reconstruct the general name of the file 2. Reconstruct the path for *V aspis* projection file 3. Import the raster and divide by 1000 4. Reconstruct the path for *V latastei* projection file 5. Import the raster and divide by 1000 6. Reconstruct the path for *V seoanei* projection file 7. Import the raster and divide by 1000 8. Create the future prediction of contact zone as `futCZ` 9. name the layer correctly with *age_ssp* 10. Stack this layer to the CZ raster.

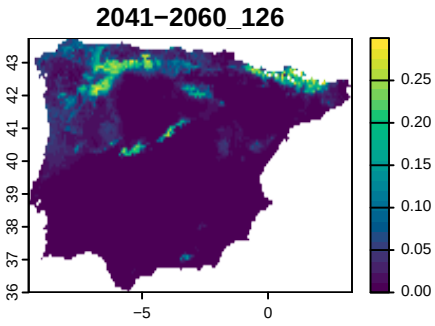
```
for (ssp in ssps) {
    for (age in ages) {
        nm <- paste0("proj_", age, "_", ssp)
        vaFile <- paste0("models/Vaspis/", nm, "/", nm, "_Vaspis_ensemble.tif")
        va <- rast(vaFile)[[1]]/1000
        vlFile <- paste0("models/Vlatastei/", nm, "/", nm, "_Vlatastei_ensemble.tif")
        vl <- rast(vlFile)[[1]]/1000
        vsFile <- paste0("models/Vseoanei/", nm, "/", nm, "_Vseoanei_ensemble.tif")
        vs <- rast(vsFile)[[1]]/1000

        # Produce Future contact zone and rename accordingly to age and ssp
        futCZ <- va*vl*vs
        names(futCZ) <- paste0(age, "_", ssp)

        # Grow CZ raster with each futCZ to store all in same file
        CZ <- c(CZ, futCZ)
    }
}
```

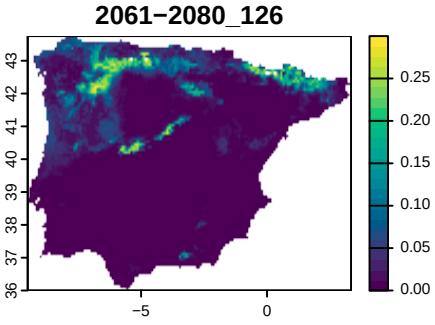
We can now plot the raster or contact zone predictions.

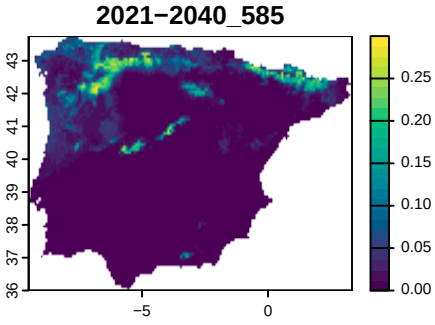
```
layout(matrix(c(1,1,1,1:7), 5, byrow=TRUE))
plot(CZ[[1]], main=names(CZ)[1], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[2]], main=names(CZ)[2], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[3]], main=names(CZ)[3], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[4]], main=names(CZ)[4], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[5]], main=names(CZ)[5], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[6]], main=names(CZ)[6], col=hcl.colors(25), range=c(0, 0.3))
plot(CZ[[7]], main=names(CZ)[7], col=hcl.colors(25), range=c(0, 0.3))
```

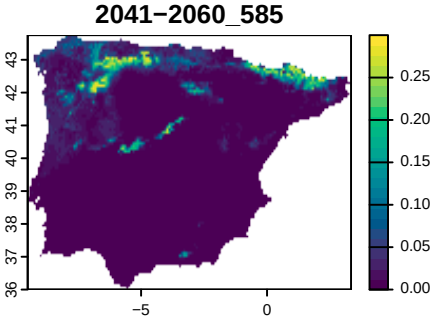

Present
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00

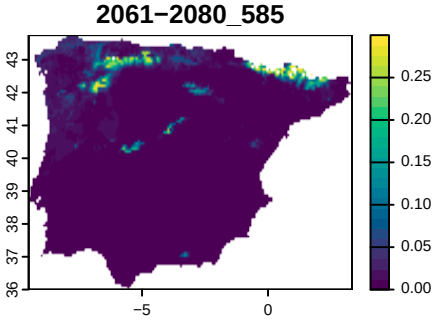

2021−2040_126
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00


2041−2060_126
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00


2061−2080_126
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00


2021−2040_585
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00


2041−2060_585
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00


2061−2080_585
43
42
41
40
39
38
37
36
−5
0
0.25
0.20
0.15
0.10
0.05
0.00

We can opt for other visualization strategies to show the change in the contact zone probabilities. For instance, we can show histograms of the probabilities to highlight how they will decrease for certain scenarios. The easiest way to do this is to extract the raster data into a data frame:

```
CZdata <- data.frame(CZ)
head(CZdata)
```

```
##       Present X2021.2040_126 X2041.2060_126 X2061.2080_126 X2021.2040_585
## 18 0.10364221             NA             NA             NA             NA
## 19 0.13685880     0.11065600     0.11286778     0.10429876     0.11304586
## 20 0.07927764     0.05262312     0.05332454     0.05201750     0.05331404
## 21 0.08235575     0.05355995     0.05383398     0.05381347     0.05567418
## 22 0.12661500     0.09097242     0.08733092     0.08592080     0.08695544
## 23 0.10871744     0.06243994     0.06434401     0.06272640     0.06621264
##    X2041.2060_585 X2061.2080_585
## 18             NA             NA
## 19     0.10022513     0.05894328
## 20     0.04383725     0.03043526
## 21     0.04631549     0.02900436
## 22     0.06917652     0.04405680
## 23     0.05170428     0.03202399
```

As the whole projected area is dominated by near-zero values (low probability of contact), we will remove these values based on a very small threshold. This will prevent the histograms from being dominated by the large number of pixels with near-zero values, providing a clearer view of the changes in contact zone probabilities.

```
CZdata[CZdata < 0.01] <- NA
```

For the histograms, we need to define the width of the bars and the range of values. In this example, the values vary between 0 and 0.28, so we choose a width of 0.01 and create a sequence of cutting points (breaks).

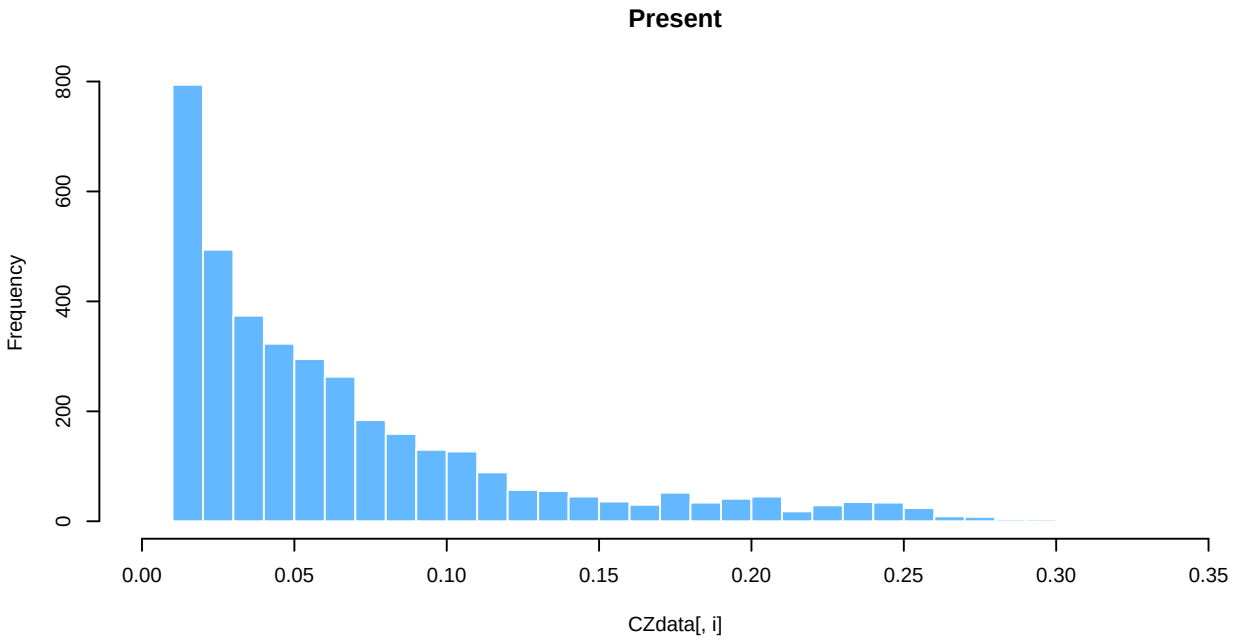
```
brk <- seq(0, 0.35, 0.01)
```

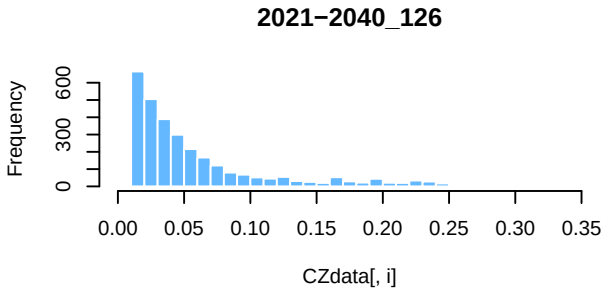
We can retrieve the names of the layers in the raster to use as titles for plotting. The order of the layers corresponds to the order of columns in the data frame.

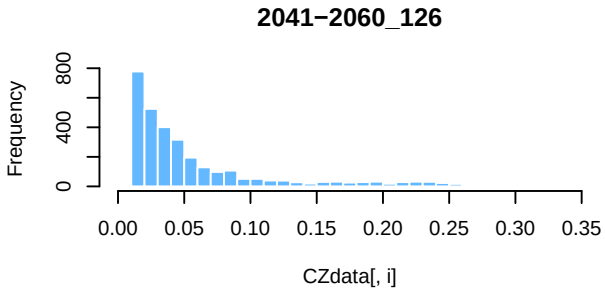
```
titles <- names(CZ)
```

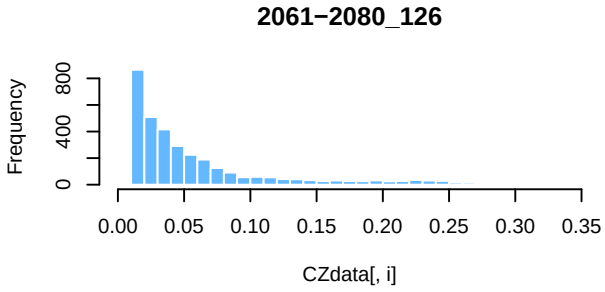
Instead of plotting each column individually, as we did above for plotting the rasters, we can use a *for* loop that iterates over the columns. We set some nice colors for the bars and plot!

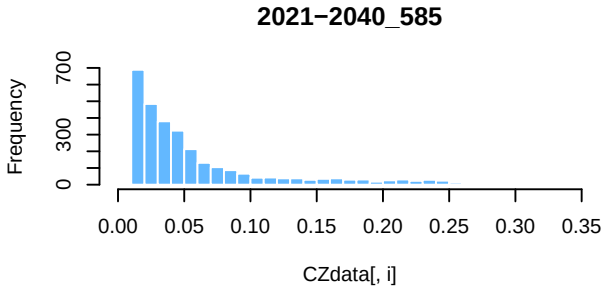
```
layout(matrix(c(1,1,1,1:7), 5, byrow=TRUE))
for (i in 1:ncol(CZdata)) {
    hist(CZdata[,i], breaks=brk, main=titles[i], col="steelblue1", border="white")
}
```

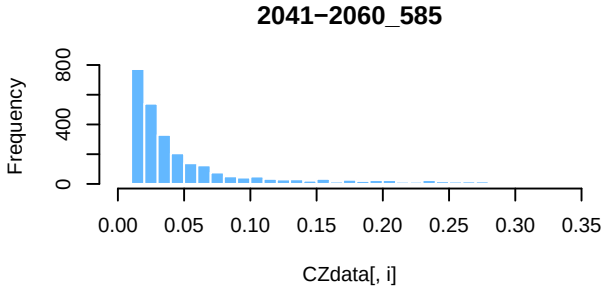


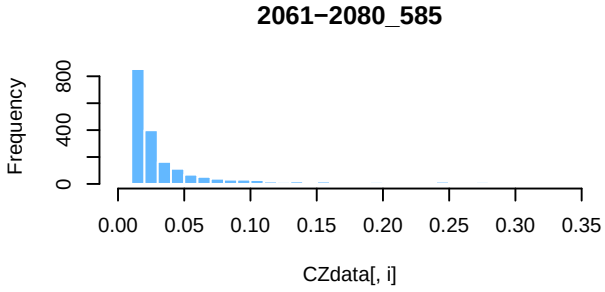












We can see that the worst-case scenario predicts a contraction of the contact zone to a narrow area by the end

of the century. Currently, this area is where all three vipers are found together, while other regions represent high potential for all species, although not all species reach those areas. One reason could be that they are at the limit of their ecological niche, and one species may be excluded by another that is more dominant in the region.

We can write the raster to a file that can be opened in any GIS for further exploration of the results.

```
{r] writeRaster(CZ, "models/sympatry.tif")
```

Conclusion

We have seen the process of ecological niche modeling using R, focusing on viper species in the Iberian Peninsula as a case study. We covered topics such as data collection, preprocessing predictors and presence data, model building, ensembling, and projection to future climate scenarios. The general steps we covered were:

- **Data Collection**: Gather presence data for the species of interest along with environmental variables like bioclimate varibles and NDVI.
- **Data Preprocessing**: Process the data by checking for duplicates in presence record and other erroneous coordinates; making sure the predictors are fully geographically align and that correlations among variables are adequate; prepare a training and projection area.
- **Model Building**: Use the *biomod2* package to build ecological niche models, selecting appropriate pseudo-absences, algorithms, and resampling strategies.
- **Model Evaluation**: Evaluate model performance using metrics like True Skill Statistic (TSS) and Receiver Operating Characteristic (ROC) curves.
- **Model Ensembling**: Ensemble multiple models to create a more robust prediction by combining different pseudo-absences sets, algorithms, and resampling strategies.
- **Projection**: Project the ensembled models to future climate scenarios to assess potential distribution changes under different climate conditions.
- **Spatial Analysis**: Analyzs spatial patterns in the model predictions, for instance, identify sympatric zones, and assess changes in contact zones over time.

At this point, you should understand that:

- Ecological niche modeling provides valuable insights into species distributions and responses to environmental changes.
- Model performance depends on various factors such as data quality, variable selection, algorithm choice, and vary between model evaluation metrics.
- Ensembling multiple models improves prediction accuracy and robustness by capturing a wider range of situtations and simplify the analyses of complex model results.
- Future climate projections highlight potential shifts in species distributions, and might provide a good insight for conservation planning and management strategies.
- Spatial analysis is an important process in the ecological modelling an, in this example, was essential to identify areas of overlap and potential interactions between species.