GAM-NICHE: Shape-Constrained GAMs to build Species Distribution Models under the ecological niche theory

AZTI

2023-05-05

Contents

About

This is a short tutorial for constructing species distribution models in R using shape-constrained generalized additive models, based on the development and application to marine fish by **?**.

The code is available in AZTI's github repository and the book is readily available here.

To cite this book, please use:

Valle, M., Citores, L., Ibaibarriaga, L., Chust, C. (2023) GAM-NICHE: Shape-Constrained GAMs to build Species Distribution Models under the ecological niche theory. https://doi.org/10.57762/fzpy-6w51

Chapter 1

Introduction

Species distribution models (SDMs) are numerical tools that combine observations of species occurrence or abundance at known locations with information on the environmental and/or spatial characteristics of those locations (**?**). SDMs are widely used as a tool for understanding species spatial ecology and are also known as ecological niche models (ENM) or habitat suitability models.

According to ecological niche theory, species response curves are unimodal with respect to environmental gradients (**?**). While a variety of statistical methods have been developed for species distribution modelling, a general problem with most of these habitat modelling approaches is that the estimated response curves can display biologically implausible shapes which do not respect ecological niche theory. This is because species response curves are fit statistically with any assumption or restriction, which sometimes do not respect the ecological niche theory. To better understand species response to environmental changes, SDMs should consider theoretical background such as the ecological niche theory and pursue the unimodality of the response curve with respect to environmental gradients.

This book provides a tutorial on how to use Shape-Constrained Generalized Additive Models (SC-GAMs) to build SDMs under the ecological niche theory framework (**?**). SC-GAMs impose monotonicity and concavity constraints in the linear predictor of the GAMs and avoid overfitting. SC-GAM is an effective alternative to fitting nonsymmetric parametric response curves, while retaining the unimodality constraint, required by ecological niche theory, for direct variables and limiting factors.

The book is organised following the key steps in good modelling practice of SDMs (**?**). First, presence data of a selected species are downloaded from GBIF/OBIS global public datasets and pseudo-absence data are created. Then, environmental data are downloaded from public repositories and extracted at each of the presence/pseudo-absence data points. Based on this dataset, an ex-

ploratory analysis is conducted to help deciding on the best modelling approach. The model is fitted to the dataset and the quality of the fit and the realism of the fitted response function are evaluated. After selecting a threshold to transform the continuous probability predictions into binary responses, the model is validated using a k-fold approach. Finally, the predicted maps are generated for visualization.

Chapter 2

Presence-absence data

In this chapter we first, download occurrence data from global open-access datasets such as Global Biodiversity Information Facility (GBIF, https://www.gbif.org/) and Ocean Biodiversity Information System (OBIS, https://obis.org/); second, clean downloaded data reformating, renaming fields and removing outliers data; and lastly, we generate a set of pseudoabsence points along our study area.

First we load a list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow
  "maptools", #plotting world map
  "ggplot2", #for plotting

  #Download presence data--------#
  "robis", # Specific library to get the occurrence data
  "rgbif",# Specific library to get the occurrence data
  "CoordinateCleaner", #to remove outlier
  "rgdal", # to work with Spatial data
  "sf", # to work with spatial data (shapefiles)
  "data.table", #for reading data,
  "dplyr", #for reading data,
  "tidyr", #for reading data
  "marmap", #bathymetry getNOAA.bathy remotes::install_github("ericpante/marmap")

  #Create pseudo-absence data--------#
  "tidyverse",
  "scales",
```

```
  "ggridges",
  "maps",     # some basic country maps
  "mapdata",   # higher resolution maps
  "mapproj",
  "mapplots",   # ICES rectangles
  "gridExtra",
  "lubridate",
  "raster" # to work with Spatial data
    )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

install_load_function(requiredPackages)
```

```
##              here        rstudioapi          maptools           ggplot2
##              TRUE              TRUE              TRUE              TRUE
##             robis             rgbif CoordinateCleaner             rgdal
##              TRUE              TRUE              TRUE              TRUE
##                sf        data.table             dplyr             tidyr
##              TRUE              TRUE              TRUE              TRUE
##            marmap         tidyverse            scales          ggridges
##              TRUE              TRUE              TRUE              TRUE
##              maps           mapdata           mapproj          mapplots
##              TRUE              TRUE              TRUE              TRUE
##         gridExtra         lubridate            raster
##              TRUE              TRUE              TRUE
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

2.1 Download presence data

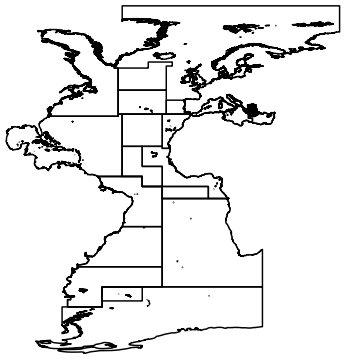
In this section we download presence data from global public datasets.

To do so, we first define our study area, in this case we select the Atlantic ocean based on the The Food and Agriculture Organization (FAO) Major Fishing Areas for Statistical Purposes and we remove Black sea subarea.

```
# url where FAO shapfile is stored
url<-"https://www.fao.org/fishery/geoserver/area/ows?service=WFS&version=1.0.0&request=GetFeature

# Download file
download.file(url,"data/spatial/FAO_AREAS.zip",mode="wb")

# Unzip downloaded file
unzip(here::here ("data", "spatial", "FAO_AREAS.zip"),exdir="data/spatial")

# Load FAO (spatial multipolygon)
FAO<- st_read(file.path("data", "spatial", "FAO_AREAS.shp"))
```

```
## Reading layer `FAO_AREAS' from data source
##   `D:\PROJECTS\github\gam-niche\data\spatial\FAO_AREAS.shp' using driver `ESRI Shapefile'
## Simple feature collection with 50 features and 15 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -85.58276 xmax: 180 ymax: 89.99
## Geodetic CRS:  WGS 84
```

```
# Select Atlantic Ocean FAO Area
FAO_Atl <- FAO[FAO$OCEAN=="Atlantic",]

# Select Black sea subarea
Black_Sea <- FAO_Atl[FAO_Atl$ID=="20",]

# Transform to sf objects
FAO_Atl.sf <- st_as_sf(FAO_Atl)
Black_Sea.sf <- st_as_sf(Black_Sea)

# Remove Black sea using st_difference (reverse of st_intersection)
FAO_Atl_no_black_sea <- st_difference(FAO_Atl.sf,Black_Sea.sf) %>%   dplyr::select (F_AREA)

# Transform to spatial polygons dataframe
study_area<- sf:::as_Spatial(FAO_Atl_no_black_sea)

plot(study_area)
```



```
# Remove unused files
rm(FAO, FAO_Atl, FAO_Atl.sf, FAO_Atl_no_black_sea, Black_Sea, Black_Sea.sf)
```

Download occurrence data from OBIS and GBIF using scientific name.

In this case we select Albacore tuna species (Thunnus alalunga).

```
# Get data from OBIS
#mydata.obis<-robis::occurrence(scientificname="Thunnus alalunga")
#save(mydata.obis, file=file.path("data","occurrences",file="mydata.obis.RData"))
load(here::here ("data", "occurrences", "mydata.obis.RData"))

# Get data from GBIF
#mydata.gbif<-occ_data(scientificName="Thunnus alalunga", hasCoordinate = TRUE, limit=
#save(mydata.gbif,file=file.path("data","occurrences",file="mydata.gbif.RData"))
load(here::here ("data", "occurrences", "mydata.gbif.RData"))
```

We now check the downloaded data and select the fields of interest.

```
# Check names for GBIF data
names(mydata.gbif)
```

```
##   [1] "key"                                 "scientificName"
##   [3] "decimalLatitude"                     "decimalLongitude"
```

```
##   [5] "issues"                    "datasetKey"
##   [7] "publishingOrgKey"          "installationKey"
##   [9] "publishingCountry"         "protocol"
##  [11] "lastCrawled"               "lastParsed"
##  [13] "crawlId"                   "hostingOrganizationKey"
##  [15] "basisOfRecord"             "occurrenceStatus"
##  [17] "taxonKey"                  "kingdomKey"
##  [19] "phylumKey"                 "orderKey"
##  [21] "familyKey"                 "genusKey"
##  [23] "speciesKey"                "acceptedTaxonKey"
##  [25] "acceptedScientificName"    "kingdom"
##  [27] "phylum"                    "order"
##  [29] "family"                    "genus"
##  [31] "species"                   "genericName"
##  [33] "specificEpithet"           "taxonRank"
##  [35] "taxonomicStatus"           "iucnRedListCategory"
##  [37] "dateIdentified"            "coordinateUncertaintyInMeters"
##  [39] "year"                      "month"
##  [41] "day"                       "eventDate"
##  [43] "modified"                  "lastInterpreted"
##  [45] "references"                "license"
##  [47] "isInCluster"               "datasetName"
##  [49] "recordedBy"                "identifiedBy"
##  [51] "geodeticDatum"             "countryCode"
##  [53] "country"                   "rightsHolder"
##  [55] "identifier"                "http://unknown.org/nick"
##  [57] "informationWithheld"       "verbatimEventDate"
##  [59] "collectionCode"            "gbifID"
##  [61] "occurrenceID"              "taxonID"
##  [63] "catalogNumber"             "institutionCode"
##  [65] "eventTime"                 "http://unknown.org/captive"
##  [67] "identificationID"          "continent"
##  [69] "stateProvince"             "verbatimLocality"
##  [71] "occurrenceRemarks"         "lifeStage"
##  [73] "datasetID"                 "eventID"
##  [75] "footprintWKT"              "originalNameUsage"
##  [77] "county"                    "identificationVerificationStatus"
##  [79] "nameAccordingTo"           "networkKeys"
##  [81] "individualCount"           "elevation"
##  [83] "waterBody"                 "institutionKey"
##  [85] "otherCatalogNumbers"       "preparations"
##  [87] "recordNumber"              "acceptedNameUsage"
##  [89] "vernacularName"            "institutionID"
##  [91] "language"                  "type"
##  [93] "identificationRemarks"     "projectId"
##  [95] "municipality"              "collectionKey"
```

```
##  [97] "higherGeography"                  "georeferenceProtocol"
##  [99] "island"                           "endDayOfYear"
## [101] "locality"                         "fieldNumber"
## [103] "startDayOfYear"                   "collectionID"
## [105] "higherClassification"             "materialSampleID"
## [107] "disposition"                      "programmeAcronym"
## [109] "organismQuantity"                 "organismQuantityType"
## [111] "samplingProtocol"                 "locationAccordingTo"
## [113] "coordinatePrecision"              "georeferencedDate"
## [115] "nomenclaturalCode"                "associatedReferences"
## [117] "taxonRemarks"                     "ownerInstitutionCode"
## [119] "bibliographicCitation"            "habitat"
## [121] "locationRemarks"                  "depth"
## [123] "http://unknown.org/license"       "taxonConceptID"
## [125] "http://unknown.org/rightsHolder"  "depthAccuracy"
## [127] "dynamicProperties"                "elevationAccuracy"
## [129] "rights"                           "georeferenceSources"
## [131] "georeferenceRemarks"              "name"
## [133] "associatedSequences"              "establishmentMeans"
## [135] "georeferenceVerificationStatus"   "accessRights"
## [137] "georeferencedBy"                  "verbatimSRS"
## [139] "previousIdentifications"          "locationID"
## [141] "acceptedNameUsageID"              "http://unknown.org/language"
## [143] "http://unknown.org/modified"      "samplingEffort"
## [145] "verbatimDepth"                    "behavior"
## [147] "eventRemarks"                     "footprintSRS"
## [149] "namePublishedInYear"              "verbatimCoordinateSystem"
## [151] "parentNameUsage"                  "http://unknown.org/taxonRankID"
## [153] "http://unknown.org/species"       "higherGeographyID"
## [155] "islandGroup"                      "organismID"
## [157] "distanceFromCentroidInMeters"     "http://unknown.org/orders"
## [159] "typeStatus"
```

```
# Select columns of interest
mydata.gbif <- mydata.gbif %>%
                dplyr::select("acceptedScientificName",
                  "decimalLongitude",
                  "decimalLatitude",
                  "year",
                  "month",
                  "day",
                  "eventDate",
                  "depth")

# Check names in for OBIS data
```

```
names(mydata.obis)
```

```
##   [1] "infraphylum"                    "country"
##   [3] "date_year"                      "scientificNameID"
##   [5] "year"                           "scientificName"
##   [7] "dropped"                        "gigaclassid"
##   [9] "aphiaID"                        "decimalLatitude"
##  [11] "subclassid"                     "gigaclass"
##  [13] "infraphylumid"                  "phylumid"
##  [15] "familyid"                       "catalogNumber"
##  [17] "basisOfRecord"                  "terrestrial"
##  [19] "id"                             "day"
##  [21] "parvphylum"                     "order"
##  [23] "dataset_id"                     "locality"
##  [25] "decimalLongitude"               "collectionCode"
##  [27] "date_end"                       "speciesid"
##  [29] "date_start"                     "month"
##  [31] "genus"                          "eventDate"
##  [33] "brackish"                       "absence"
##  [35] "subfamily"                      "genusid"
##  [37] "originalScientificName"         "marine"
##  [39] "subphylumid"                    "subfamilyid"
##  [41] "institutionCode"                "date_mid"
##  [43] "class"                          "orderid"
##  [45] "waterBody"                      "kingdom"
##  [47] "classid"                        "phylum"
##  [49] "species"                        "subphylum"
##  [51] "subclass"                       "family"
##  [53] "category"                       "kingdomid"
##  [55] "parvphylumid"                   "node_id"
##  [57] "flags"                          "sss"
##  [59] "shoredistance"                  "sst"
##  [61] "bathymetry"                     "maximumDepthInMeters"
##  [63] "minimumDepthInMeters"           "sex"
##  [65] "depth"                          "coordinatePrecision"
##  [67] "verbatimCoordinates"            "occurrenceRemarks"
##  [69] "individualCount"                "occurrenceStatus"
##  [71] "modified"                       "materialSampleID"
##  [73] "occurrenceID"                   "scientificNameAuthorship"
##  [75] "taxonRank"                      "datasetID"
##  [77] "associatedReferences"           "bibliographicCitation"
##  [79] "coordinateUncertaintyInMeters"  "vernacularName"
##  [81] "footprintWKT"                   "specificEpithet"
##  [83] "recordNumber"                   "georeferenceRemarks"
##  [85] "stateProvince"                  "continent"
```

```
##  [87] "recordedBy"                    "rightsHolder"
##  [89] "language"                      "type"
##  [91] "license"                       "ownerInstitutionCode"
##  [93] "datasetName"                   "geodeticDatum"
##  [95] "accessRights"                  "dynamicProperties"
##  [97] "county"                        "samplingEffort"
##  [99] "lifeStage"                     "footprintSRS"
## [101] "samplingProtocol"              "parentEventID"
## [103] "eventID"                       "eventRemarks"
## [105] "identifiedBy"                  "georeferencedBy"
## [107] "minimumElevationInMeters"      "maximumElevationInMeters"
## [109] "dateIdentified"                "behavior"
## [111] "georeferenceProtocol"          "verbatimDepth"
## [113] "organismQuantity"              "organismQuantityType"
## [115] "startDayOfYear"                "collectionID"
## [117] "typeStatus"                    "institutionID"
## [119] "acceptedNameUsage"             "preparations"
## [121] "countryCode"                   "otherCatalogNumbers"
## [123] "references"                    "fieldNumber"
## [125] "endDayOfYear"                  "eventTime"
## [127] "locationID"                    "taxonRemarks"
## [129] "georeferencedDate"             "verbatimEventDate"
## [131] "identificationRemarks"         "nomenclaturalCode"
## [133] "taxonomicStatus"               "higherClassification"
## [135] "higherGeography"               "verbatimLatitude"
## [137] "verbatimLongitude"             "establishmentMeans"
## [139] "verbatimLocality"              "georeferenceVerificationStatus"
## [141] "island"                        "identificationQualifier"
## [143] "identificationID"              "informationWithheld"
## [145] "islandGroup"                   "acceptedNameUsageID"
## [147] "locationRemarks"               "verbatimSRS"
## [149] "previousIdentifications"
```

```
# Select columns of interest
mydata.obis <-  mydata.obis %>%
                dplyr::select("scientificName",
                  "decimalLongitude",
                  "decimalLatitude",
                  "date_year",
                  "month",
                  "day",
                  "eventDate",
                  "depth",
                  "bathymetry",
                  "occurrenceStatus",
```

```
                    "sst")
```

Reformat the data adding a new field and renaming some columns from mydata.gbif dataframe in order to have the same columns and be able to join both downloaded datasets.

```
mydata.gbif <- mydata.gbif %>%
    dplyr::rename(scientificName= "acceptedScientificName") %>%
    dplyr::rename(date_year = "year") %>%
    dplyr::mutate(bathymetry= NA) %>%
    dplyr::mutate(occurrenceStatus=1) %>%
    dplyr::mutate(sst= NA)

# Join data from OBIS and GBIF
mydata.fus<-rbind(mydata.obis,mydata.gbif)

# Assign unique scientific name
mydata.fus <- mydata.fus %>%
    dplyr::mutate(scientificName= paste(mydata.obis$scientificName[1]))

# Remove unused files
rm(mydata.gbif, mydata.obis)
```

We now clean downloaded raw data.

```
# Give date format to eventDate and fill out month and date_year columns
  mydata.fus$eventDate <- as.Date(mydata.fus$eventDate)
  mydata.fus$date_year <- as.numeric(mydata.fus$date_year)
  mydata.fus$month <- as.numeric(mydata.fus$month)

# Mutate occurrenceStatus column giving them a value of 1
unique (mydata.fus$occurrenceStatus)
```

```
## [1] NA        "present" "P"       "Present" "NA"      "1"
```

```
# Assign 1 value to all occurrences
mydata.fus <- mydata.fus  %>%
    dplyr::mutate(occurrenceStatus = 1)
```

We remove outliers based on distance method (total distance= 1000 km).

```
out.dist <- cc_outl(x=mydata.fus,
                lon = "decimalLongitude", lat = "decimalLatitude",
```

```
                species = "scientificName",
                method="distance", tdi=1000, # distance method with tdi=1000km
                thinning=T, thinning_res=0.5,
                value="flagged")

# Remove outliers from the data
mydata.fus <- mydata.fus[out.dist, ]
```

Remove duplicates.

```
# First create a vector containing longitude, latitude and event date information
date <- cbind(mydata.fus$decimalLongitude,mydata.fus$decimalLatitude,mydata.fus$eventDa

# Remove the duplicated records
mydata.fus<-mydata.fus[!duplicated(date),]

# Remove unused files
rm(date)
```

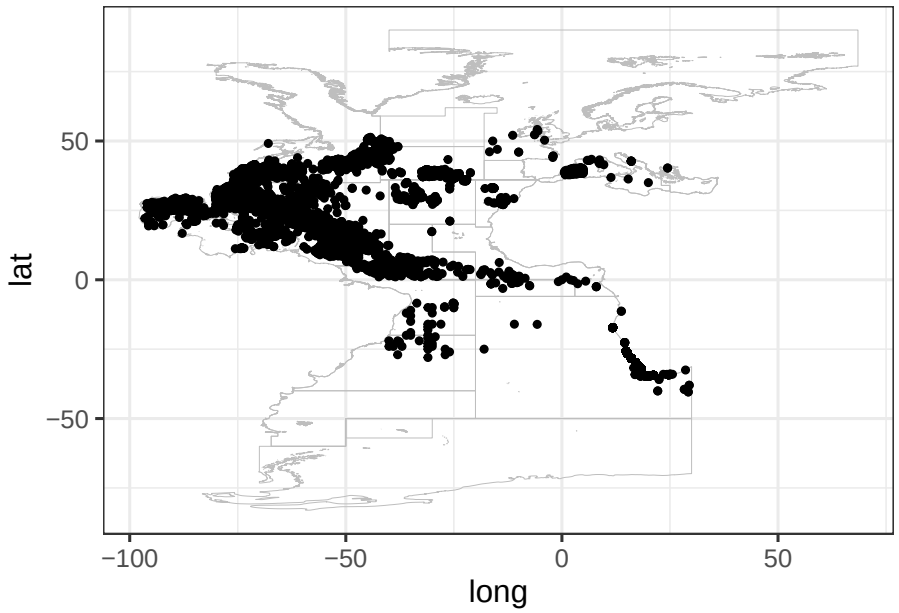
Mask retrieved occurrence data to our study area and add bathymetry value to each occurrence point.

```
# Assign coordinate format and projection to be able to use FAO Atlantic as a mask
dat <- data.frame(cbind(mydata.fus$decimalLongitude,mydata.fus$decimalLatitude))
ptos<-as.data.table(dat,keep.columnnames=TRUE)

coordinates(ptos) <- ~ X1 + X2

# Assign projection
proj4string(ptos) <-proj4string(study_area)

# Select only occurrences from FAO Atlantic

match2<-data.frame(subset(mydata.fus,!is.na(over(ptos, study_area)[,1])))

# Extract the FAO area of each point
match3<-data.frame(subset(over(ptos, study_area), !is.na(over(ptos, study_area)[,1])))

# Create data frame with area, name, long, lat and year
df0<-cbind(F_AREA=match3$F_AREA,match2)[,c("F_AREA","scientificName","decimalLongitude"

# Rename some columns
names(df0)[3:5]<-c("LON","LAT","YEAR")

# Add bathymetry from NOAA
```

```
#bathy <- marmap::getNOAA.bathy(lon1=-100,lon2=30,lat1=-41,lat2=55, resolution = 1,
load(here::here ("data", "spatial", "bathy.RData"))

df0$bathymetry <- get.depth(bathy, df0[,c("LON","LAT")], locator=F)$depth

# Remove unused files
rm(mydata.fus, match2, match3, dat, ptos)
```

We plot the occurrence data.

```
ggplot() +
   geom_path(data = study_area,
             aes(x = long, y = lat, group = group),
             color = 'gray', linewidth = .2) +
   geom_point(data=df0, aes(x=LON, y=LAT))
```



And, we save the data.

```
save(df0, file = "data/occurrences/occ.RData")
save(study_area, file = "data/spatial/study_area.RData")
```

2.2 Create pseudo-absence data

After saving presence data for the species of interest we need to generate absence data (with a constant prevalence of 50%, **?**,**?**), in order to work with logistic regression SDM afterwards. For that, we generate a buffer around each presence data point, with a radius of 100km, where no absences can be generated.

Before starting the pseudo-absence generation process, we delete observations in land (positive bathymetry) and select data between 2000 and 2014 (same temporal range as the environmental data that we will load later).

```
#remove points in land
df0<-subset(df0,bathymetry<0)

#use years from 2000 to 2014
df0<- subset(df0, YEAR<=2014 & YEAR>=2000)
```

Then we transform the presence data frame to "SpatialPointsDataFrame" class object and the to "sf" class object so we can operate and plot easily with spatial data. We can see the characteristics of this object through its summary.

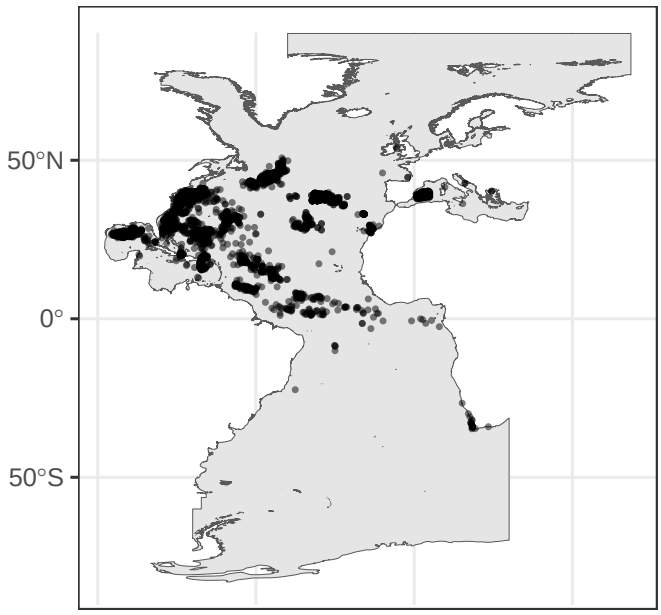
```
#convert to spatial point data frame
df<-df0 ; coordinates(df)<- ~LON+LAT
crs(df)<-crs(study_area)

#convert to sf
study_area.sf<-st_union(st_as_sf(study_area))
df.sf<-st_as_sf(df)

summary(df)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## LON -95.65 24.45000
## LAT -34.71 54.15014
## Is projected: FALSE
## proj4string : [+proj=longlat +datum=WGS84 +no_defs]
## Number of points: 14903
## Data attributes:
##     F_AREA          scientificName          YEAR      occurrenceStatus
##  Length:14903       Length:14903       Min.   :2000   Min.   :1
##  Class :character   Class :character   1st Qu.:2001   1st Qu.:1
##  Mode  :character   Mode  :character   Median :2002   Median :1
##                                        Mean   :2003   Mean   :1
```

```
##                                          3rd Qu.:2004   3rd Qu.:1
##                                          Max.   :2013   Max.   :1
##    bathymetry
##  Min.   :-8404.09
##  1st Qu.:-1916.92
##  Median :-1030.77
##  Mean   :-1433.61
##  3rd Qu.: -351.32
##  Max.   :   -1.13
```

We can easily plot sf objects using ggplot, here we plot the area of study and the presence data points.

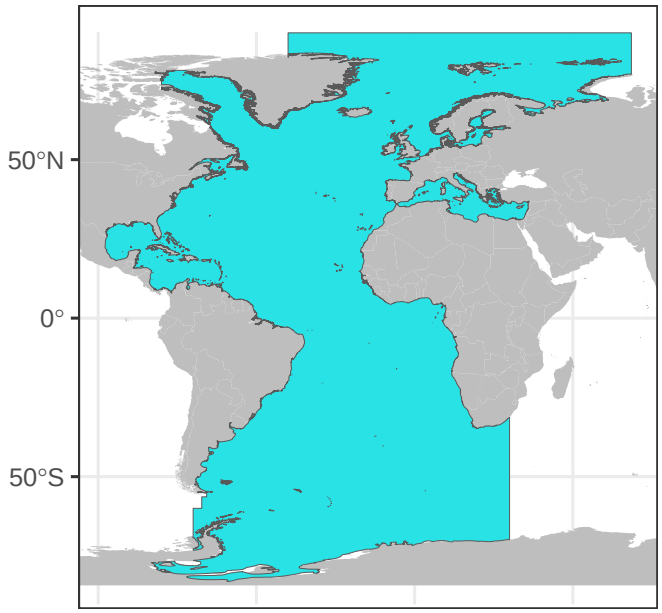
```
ggplot(study_area.sf) +
  geom_sf() +
  geom_sf(data=st_union(df.sf),
          size=1,
          alpha=0.5)
```



We save a base plot (p0) with the world map and our area of study in blue.

```
# basic ggplot
global <- map_data("worldHires")
```

```
p0 <- ggplot() +
  annotation_map(map=global, fill="grey")+
  geom_sf(data=study_area.sf,fill=5)

print(p0)
```



In order to generate a buffer around each occurrence data point, we need to work with euclidean distances, so first, we need to transform the decimal latitude and longitude values to UTM.

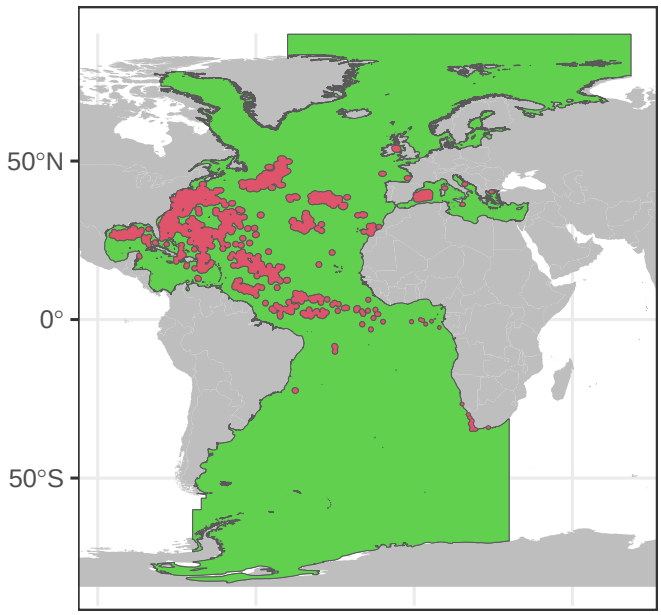
```
# function to find your UTM.
lonlat2UTM = function(lonlat) {
  utm = (floor((lonlat[1] + 180) / 6) %% 60) + 1
  if(lonlat[2] > 0) {
    utm + 32600
  } else{
    utm + 32700
  }
}

(EPSG_2_UTM <- lonlat2UTM(c(mean(df$LON), mean(df$LAT))))
```

```
## [1] 32623
```

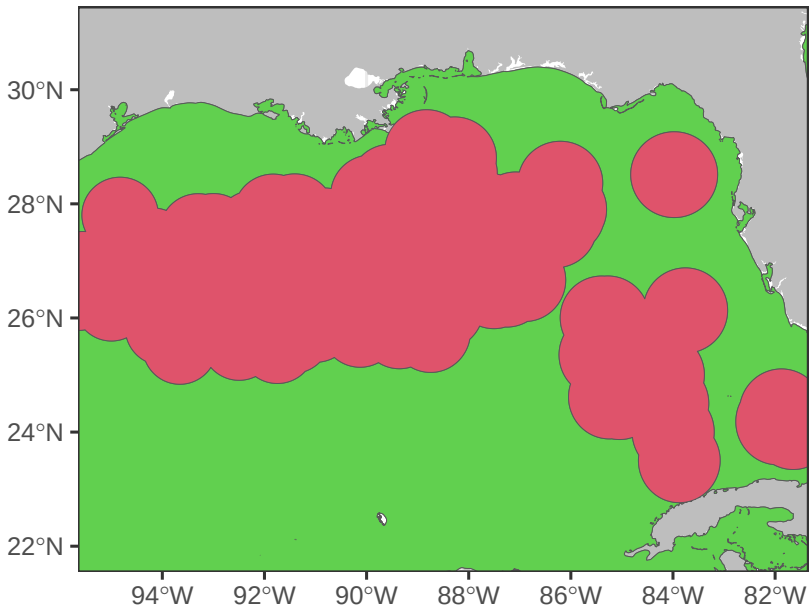
```
# transform study_area and data points to UTMs (in m)
aux <- st_transform(study_area.sf, EPSG_2_UTM)
df.sf.utm <- st_transform(df.sf, EPSG_2_UTM)
```

Now, we can create buffers of 100 km around the points and join the resulting polygons. Then this buffer is intersected with the area of study defining the area where the pseudo-absences can be generated. To visualize the defined areas, we plot the buffers in red and the area that we will use to generate pseudo-absences in green.

```
# create buffers of 100000m
buffer <- st_buffer(df.sf.utm, dist=100000)
buffer <- st_union(buffer)

# intersect the are with the buffer
aux0 <- st_difference(aux, buffer)

# ggplot for all data
p0 +
  geom_sf(data=aux0,fill=3) +
  geom_sf(data=buffer,fill=2)
```



```
#zoom
p0 +
  geom_sf(data=aux0,fill=3) +
  geom_sf(data=buffer,fill=2)+
  coord_sf(xlim=c(-95,-82), ylim=c(22,31))
```



We create a data frame for pseudo-absences with the same dimensions as the presences data frame.

```
# Generate the pseudo-absence data frame
pseudo <- matrix(data=NA, nrow=dim(df0)[1], ncol=dim(df0)[2])
pseudo <- data.frame(pseudo)
names(pseudo) <- names(df0)
```

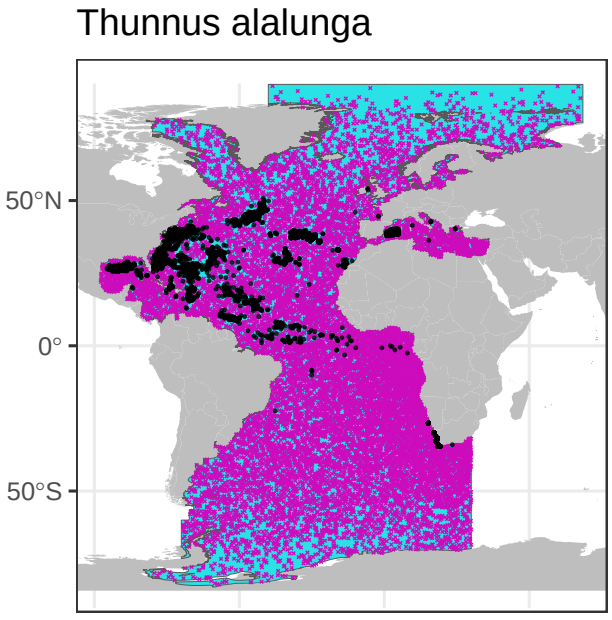
To generate the pseudo-absence data points, we sample randomly from the defined area and we extract their latitude and longitude to incorporate them in the created data frame. We set the occurrenceStatus equal to 0 as they are absences.

```
# set the seed
set.seed(1)

#sample from the defined area
rp.sf <- st_sample(aux0, size=dim(df.sf.utm)[1], type="random") # randomly sample poin
```

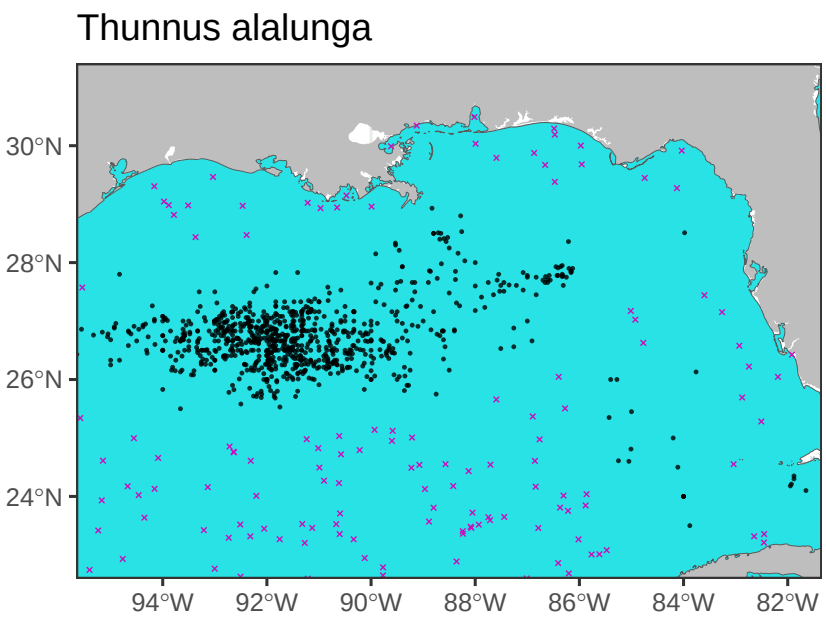
```
# transform to lat and lon and extract coordinates as data.frame
rp.sf <- st_transform(rp.sf, 4326)
rp <- as.data.frame(st_coordinates(rp.sf))
pseudo$LON <- rp$X
pseudo$LAT <- rp$Y

# complete the rest of columns
pseudo$scientificName <- df0$scientificName
pseudo$occurrenceStatus  <- 0
```

We can plot the generated pseudo-absence data (in pink) in the map, together with the presence data points (in black).

```
p0 +
  geom_sf(data=rp.sf, col=6, shape=4,size=0.5)+
  geom_sf(data=df.sf.utm, col=1, alpha=0.8,size=0.5)+
  ggtitle(unique(df$scientificName))
```




```
#zoom
p0 +
  geom_sf(data=rp.sf, col=6, shape=4,size=1)+
  geom_sf(data=df.sf.utm, col=1, alpha=0.8,size=0.5)+
  coord_sf(xlim=c(-95,-82), ylim=c(23,31))+
  ggtitle(unique(df$scientificName))
```



Finally we join the presence and pseudo-absence data frames selecting the columns of interest and save the new data frame as well as the area object.

```
# Join the two data sets
PAdata <- rbind(df0, pseudo)[,c("scientificName","LON","LAT","YEAR","occurrenceStatus"

# Save the final dataset of occurrence and pseudo-absence points
save(list=c("PAdata"),file=file.path("data","outputs_for_modelling",file="PAdata.RData
```

Chapter 3

Environmental data

In this chapter we first, download environmental data from a public repository; second, we crop the data to our area of interest, and we save it as raster stack.

As in chapter 2, first, we load a list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow
  "maptools", #plotting world map
  "ggplot2", #for plotting
  "knitr",  # format tables
  "kableExtra", # format tables
  "raster", # to work with spatial data
  "dplyr",

  #DOWNLOAD FROM PUBLIC REPOSITORIES --------#
  "sdmpredictors" #to access Bio-ORACLE dataset

    )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
```

dataset_code	terrestrial	marine	url	description
WorldClim	TRUE	FALSE	http://www.worldclim.org/	WorldClim is a set of glo
Bio-ORACLE	FALSE	TRUE	https://bio-oracle.org/	Bio-ORACLE is a set of
MARSPEC	FALSE	TRUE	http://marspec.org/	MARSPEC is a set of hig
ENVIREM	TRUE	FALSE	https://envirem.github.io/	The ENVIREM dataset i
Freshwater	TRUE	FALSE	https://www.earthenv.org/streams	The dataset consists of n

```
}

install_load_function(requiredPackages)
```

```
##          here    rstudioapi      maptools       ggplot2         knitr
##          TRUE          TRUE          TRUE          TRUE          TRUE
##    kableExtra        raster         dplyr sdmpredictors
##          TRUE          TRUE          TRUE          TRUE
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

3.1 Download from public repositories

Environmental data can be available from different sources. In this case, we used the Bio-ORACLE (ocean rasters for analyses of climate and environment) database (**??**). These data are publicly available and are easily accessible from the `sdmpredictors` package `sdmpredictors`.

We can check the list of available datasets with the function `list_datasets` from `sdmpredictors` package, as follows:

```
mydat <- list_datasets()

kable(mydat)%>%
  kable_styling("striped") %>%
  scroll_box(height="600px", width = "100%")
```

By default this function returns all the supported datasets. To return only marine datasets we can set the `marine` argument equal to `TRUE` or equivalently we could set the `terrestrial` and `freshwater` arguments equal to `FALSE`:

```
mydat <- list_datasets(marine=T)
# or equivalently:
# mydat <- list_datasets(terrestrial=F, freshwater=F)
# mydat <- list_datasets(marine=T, terrestrial=F, freshwater=F)
```

There are two datasets (Bio-ORACLE and MARSPEC) that have marine data. The function `list_layers` returns information on the layers of one or more datasets. So, we can see the layers available in the Bio-ORACLE dataset as follows:

```
mytab <- list_layers("Bio-ORACLE")

kable(mytab)%>%
  kable_styling("striped") %>%
  scroll_box(height="600px", width = "100%")
```

Once we identify the dataset and the layers we are interested on, we can extract their details from the list. In this case, we download data on chlorophyll, salinity, diffuse attenuation coefficient and sea surface temperature.

```
target <- c("BO2_chlomean_ss",
            "BO2_salinitymean_ss",
            "BO_damean",
            "BO_sstmean")

# Extrat details from the list
myvars <- mytab %>%
  dplyr::filter (mytab$layer_code %in% target)

myvars$name
```

```
## [1] "Diffuse attenuation coefficient at 490 nm (mean)"
## [2] "Sea surface temperature (mean)"
## [3] "Chlorophyll concentration (mean)"
## [4] "Sea surface salinity (mean)"
```

And we can download them using the function `load_layers`

```
# Download layers
myBioracle.layers <- load_layers(c("BO2_chlomean_ss", "BO2_salinitymean_ss", "BO_damean" ,"BO_sst
```

Note that the resulting object is a `rasterStack`, where each variable is a layer

	dataset_code	layer_code	name
74	Bio-ORACLE	BO_cloudmax	Cloud fraction (maximum)
452	Bio-ORACLE	BO22_calcite	Calcite (mean)
453	Bio-ORACLE	BO22_cloudmax	Cloud fraction (maximum)
454	Bio-ORACLE	BO22_cloudmean	Cloud fraction (mean)
455	Bio-ORACLE	BO22_cloudmin	Cloud fraction (minimum)
456	Bio-ORACLE	BO22_damax	BO_damax
457	Bio-ORACLE	BO22_damean	BO_damean
458	Bio-ORACLE	BO22_damin	BO_damin
459	Bio-ORACLE	BO22_parmax	BO_parmax
460	Bio-ORACLE	BO22_parmean	BO_parmean
461	Bio-ORACLE	BO22_ph	BO_ph
462	Bio-ORACLE	BO_calcite	Calcite (mean)
463	Bio-ORACLE	BO_chlomax	Chlorophyll A (maximum)
464	Bio-ORACLE	BO_chlomean	Chlorophyll A (mean)
465	Bio-ORACLE	BO_chlomin	Chlorophyll A (minimum)
466	Bio-ORACLE	BO_chlorange	Chlorophyll A (range)
467	Bio-ORACLE	BO_cloudmean	Cloud fraction (mean)
468	Bio-ORACLE	BO_cloudmin	Cloud fraction (minimum)
469	Bio-ORACLE	BO_damax	Diffuse attenuation coefficient at 490 nm
470	Bio-ORACLE	BO_damean	Diffuse attenuation coefficient at 490 nm
471	Bio-ORACLE	BO_damin	Diffuse attenuation coefficient at 490 nm
472	Bio-ORACLE	BO_dissox	Dissolved oxygen
473	Bio-ORACLE	BO_nitrate	Nitrate
474	Bio-ORACLE	BO_parmax	Photosynthetically available radiation (r
475	Bio-ORACLE	BO_parmean	Photosynthetically available radiation (r
476	Bio-ORACLE	BO_ph	pH
477	Bio-ORACLE	BO_phosphate	Phosphate
478	Bio-ORACLE	BO_salinity	Salinity
479	Bio-ORACLE	BO_silicate	Silicate
480	Bio-ORACLE	BO_sstmax	Sea surface temperature (maximum)
481	Bio-ORACLE	BO_sstmean	Sea surface temperature (mean)
482	Bio-ORACLE	BO_sstmin	Sea surface temperature (minimum)
483	Bio-ORACLE	BO_sstrange	Sea surface temperature (range)
484	Bio-ORACLE	BO_bathymin	Bathymetry (minimum)
485	Bio-ORACLE	BO_bathymax	Bathymetry (maximum)
486	Bio-ORACLE	BO_bathymean	Bathymetry (mean)
487	Bio-ORACLE	BO2_chlomax_bdmax	Chlorophyll concentration (maximum at
488	Bio-ORACLE	BO2_chlomax_bdmean	Chlorophyll concentration (maximum at
489	Bio-ORACLE	BO2_chlomax_bdmin	Chlorophyll concentration (maximum at
490	Bio-ORACLE	BO2_chlomean_bdmax	Chlorophyll concentration (mean at max
491	Bio-ORACLE	BO2_chlomean_bdmean	Chlorophyll concentration (mean at mea
492	Bio-ORACLE	BO2_chlomean_bdmin	Chlorophyll concentration (mean at min
493	Bio-ORACLE	BO2_chlomin_bdmax	Chlorophyll concentration (minimum at
494	Bio-ORACLE	BO2_chlomin_bdmean	Chlorophyll concentration (minimum at
495	Bio-ORACLE	BO2_chlomin_bdmin	Chlorophyll concentration (minimum at
496	Bio-ORACLE	BO2_chlorange_bdmax	Chlorophyll concentration (range at max
497	Bio-ORACLE	BO2_chlorange_bdmean	Chlorophyll concentration (range at mea
498	Bio-ORACLE	BO2_chlorange_bdmin	Chlorophyll concentration (range at min
499	Bio-ORACLE	BO2_chloltmax_bdmax	Chlorophyll concentration (longterm ma
500	Bio-ORACLE	BO2_chloltmax_bdmean	Chlorophyll concentration (longterm ma
501	Bio-ORACLE	BO2_chloltmax_bdmin	Chlorophyll concentration (longterm ma
502	Bio-ORACLE	BO2_chloltmin_bdmax	Chlorophyll concentration (longterm mi
503	Bio-ORACLE	BO2_chloltmin_bdmean	Chlorophyll concentration (longterm ma

```
class(myBioracle.layers)
```

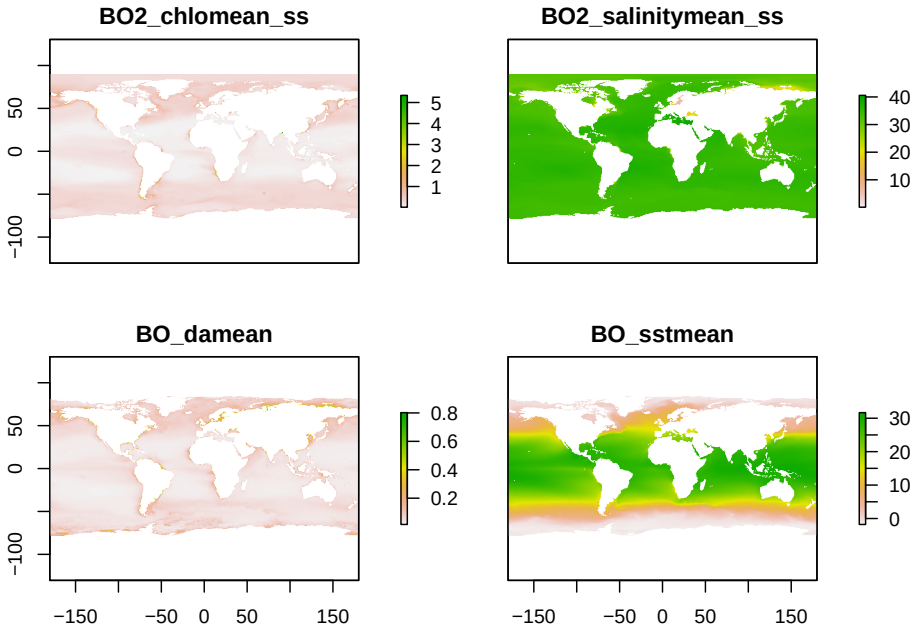
```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

```
myBioracle.layers
```

```
## class      : RasterStack
## dimensions : 2160, 4320, 9331200, 4  (nrow, ncol, ncell, nlayers)
## resolution : 0.08333333, 0.08333333  (x, y)
## extent     : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## names      : BO2_chlomean_ss, BO2_salinitymean_ss, BO_damean, BO_sstmean
## min values :        0.015806,            0.059304,  0.012000,  -1.801000
## max values :        5.394896,           40.651300,  0.972000,  32.918000
```

And we can plot it:

```
raster::plot(myBioracle.layers)
```



Note that the functions `dataset_citations` and `layer_citations` provide the bibliographic entries of the datasets and layers for proper citation:

```
print(dataset_citations("Bio-ORACLE"))
```

```
## [1] "Assis J, Tyberghein L, Bosch S, Heroen V, Serrão E, De Clerck O, Tittensor D (2
## [2] "Tyberghein L, Heroen V, Pauly K, Troupin C, Mineur F, De Clerck O (2012). \"Bio
```

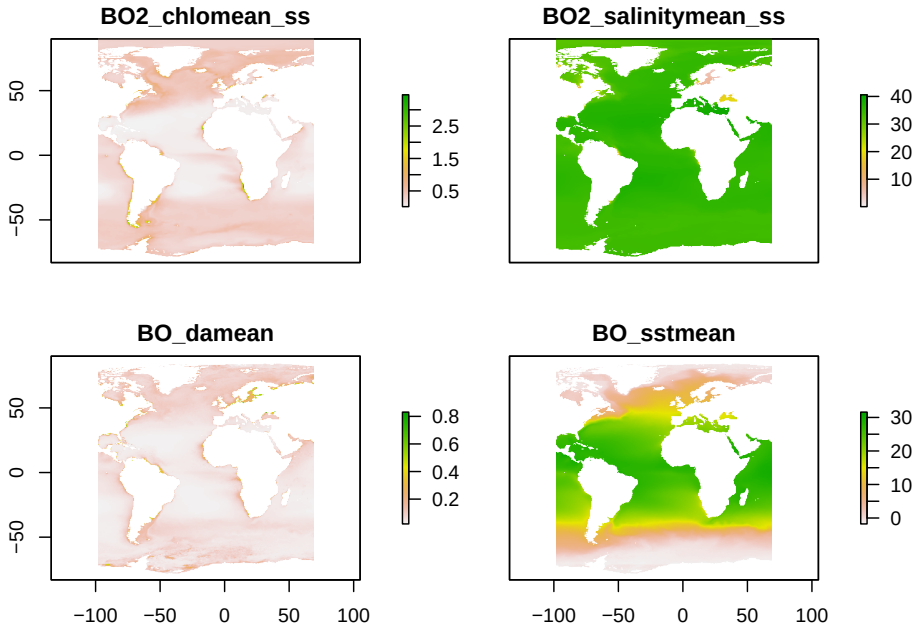
```
print(layer_citations("BO2_chlomean_ss"))
```

```
## [1] "Assis J, Tyberghein L, Bosch S, Heroen V, Serrão E, De Clerck O, Tittensor D (2
## [2] "Tyberghein L, Heroen V, Pauly K, Troupin C, Mineur F, De Clerck O (2012). \"Bio
```

In case we are not interested on the whole area, we can crop the raster objects to the area of interest.

For example, we can load the `study_area` object that is a SpatialPolygonsDataFrame that has been created previously and defines the extent of our spatial data and we can crop the rasterStack to the same extent:

```
load(here::here ("data", "spatial", "study_area.RData"))

mylayers <- crop(myBioracle.layers, extent(study_area))

plot(mylayers)
```



To facilitate subsequent access, the rasterStack with the downloaded data is saved in a local folder:

```
writeRaster(mylayers, filename="data/env/mylayers.tif", options="INTERLEAVE=BAND", overwrite=TRUE
```

3.2 Prepare the final dataset

In this chapter we first, extract environmental data associated to the presence-pseudoabsence data, we explore the data we got, we check correlation between variables and we calculate the Variance Inflation Factor (VIF) to make a selection of the variables we are going to use in the model.

First we load a list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow

  #EXTRACT ENVIRONMENTAL DATA AND PLOTS
  "sp", # spatial data
  "raster", #spatial data
  "dplyr",
  "tidyr",
  "ggplot2",
  "ggcorrplot",

  #CORRELATION ANALYSIS
  "GGally", #correlation analysis
  "HH" #calculate VIF
    )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

install_load_function(requiredPackages)
```

```
##       here rstudioapi         sp     raster      dplyr      tidyr    ggplot2
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
```

```
## ggcorrplot     GGally         HH
##       TRUE       TRUE       TRUE
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

3.3 Extract environmental data associated to presence-absence data

Once we have prepared our species distribution data (occurrences and pseudo-absences) and the environmental rasters, we need to merge both sources of data. First, we load the objects created in previous sections:

```
#load presence-absence data
load(here::here ("data", "outputs_for_modelling", "PAdata.RData"))

# load environmental rasters
mylayers<-stack(here::here ("data", "env", "mylayers.tif"))
```

Now we can extract the environmental data associated to each of the species data points using the function `extract` from the `raster` package. The method employed is `bilinear` that returns the interpolated value from the four nearest raster cells.

```
raster_ex <- raster::extract(x=mylayers, y=PAdata[,c("LON","LAT")], method="bilinear",

colnames(raster_ex)[-1]<-c("BO2_chlomean_ss", "BO2_salinitymean_ss", "BO_damean" ,"BO_s

head(raster_ex)
```

```
##   ID BO2_chlomean_ss BO2_salinitymean_ss BO_damean BO_sstmean
## 1  1      1.20410309            35.42666 0.1973854   16.99439
## 2  2      1.20410309            35.42666 0.1973854   16.99439
## 3  3      0.08455528            36.43070 0.0290000   25.52963
## 4  4      1.20410309            35.42666 0.1973854   16.99439
## 5  5      0.72283228            34.20426 0.0817736   15.21486
## 6  6      1.20410309            35.42666 0.1973854   16.99439
```

We merge the presence/absence data and the environmental data:

```
data <- cbind(PAdata, raster_ex)
```

We can conduct some quick checks on the new dataset:

```
dim(data)
```

```
## [1] 29806    10
```

```
str(data)
```

```
## 'data.frame':    29806 obs. of  10 variables:
##  $ scientificName     : chr  "Thunnus alalunga" "Thunnus alalunga" "Thunnus alalunga" "Thunnus
##  $ LON                : num  18.5 18.5 -76.6 18.5 -69.3 ...
##  $ LAT                : num  -34.4 -34.4 28.4 -34.4 39.9 ...
##  $ YEAR               : num  2004 2004 2000 2000 2000 ...
##  $ occurrenceStatus   : num  1 1 1 1 1 1 1 1 1 ...
##  $ ID                 : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ BO2_chlomean_ss    : num  1.2041 1.2041 0.0846 1.2041 0.7228 ...
##  $ BO2_salinitymean_ss: num  35.4 35.4 36.4 35.4 34.2 ...
##  $ BO_damean          : num  0.1974 0.1974 0.029 0.1974 0.0818 ...
##  $ BO_sstmean         : num  17 17 25.5 17 15.2 ...
```

```
head(data)
```

```
##      scientificName      LON      LAT YEAR occurrenceStatus ID BO2_chlomean_ss
## 1  Thunnus alalunga  18.4972 -34.3569 2004                 1  1      1.20410309
## 5  Thunnus alalunga  18.4972 -34.3569 2004                 1  2      1.20410309
## 8  Thunnus alalunga -76.6000  28.4000 2000                 1  3      0.08455528
## 10 Thunnus alalunga  18.4972 -34.3569 2000                 1  4      1.20410309
## 11 Thunnus alalunga -69.3100  39.8800 2000                 1  5      0.72283228
## 12 Thunnus alalunga  18.4972 -34.3569 2001                 1  6      1.20410309
##    BO2_salinitymean_ss BO_damean BO_sstmean
## 1             35.42666 0.1973854   16.99439
## 5             35.42666 0.1973854   16.99439
## 8             36.43070 0.0290000   25.52963
## 10            35.42666 0.1973854   16.99439
## 11            34.20426 0.0817736   15.21486
## 12            35.42666 0.1973854   16.99439
```

```
summary(data)
```

```
##  scientificName          LON               LAT               YEAR
```

```
##  Length:29806       Min.   :-97.805   Min.   :-82.187   Min.   :2000
##  Class :character   1st Qu.:-69.221   1st Qu.:-29.933   1st Qu.:2001
##  Mode  :character   Median :-28.967   Median : 24.006   Median :2002
##                     Mean   :-31.118   Mean   :  7.678   Mean   :2003
##                     3rd Qu.:  2.933   3rd Qu.: 38.250   3rd Qu.:2004
##                     Max.   : 67.648   Max.   : 89.418   Max.   :2013
##                                                         NA's   :14903
##  occurrenceStatus       ID         BO2_chlomean_ss   BO2_salinitymean_ss
##  Min.   :0.0      Min.   :    1   Min.   :0.01585   Min.   : 0.1632
##  1st Qu.:0.0      1st Qu.: 7452   1st Qu.:0.08960   1st Qu.:34.5032
##  Median :0.5      Median :14904   Median :0.24648   Median :35.4267
##  Mean   :0.5      Mean   :14904   Mean   :0.42505   Mean   :35.2388
##  3rd Qu.:1.0      3rd Qu.:22355   3rd Qu.:0.68378   3rd Qu.:36.2156
##  Max.   :1.0      Max.   :29806   Max.   :3.44767   Max.   :39.2467
##                                   NA's   :98        NA's   :98
##    BO_damean         BO_sstmean
##  Min.   :0.02180   Min.   :-1.705
##  1st Qu.:0.03505   1st Qu.:14.852
##  Median :0.05039   Median :18.898
##  Mean   :0.07833   Mean   :18.117
##  3rd Qu.:0.08800   3rd Qu.:25.373
##  Max.   :0.69655   Max.   :30.663
##  NA's   :139       NA's   :139
```

The new dataset has 29806 rows and 10 columns, and there are 139 NA's in the environmental dataset. We remove the points with NA's:

```
data <- data %>%
  dplyr::select (-YEAR) %>% #we remove year column because pseudoabsences miss this in
  na.omit()
```

We check again the dataset:

```
dim(data)
```

```
## [1] 29661     9
```

```
summary(data)
```

```
##  scientificName          LON               LAT          occurrenceStatus
##  Length:29661       Min.   :-97.805   Min.   :-77.508   Min.   :0.0000
##  Class :character   1st Qu.:-69.330   1st Qu.:-29.933   1st Qu.:0.0000
##  Mode  :character   Median :-28.946   Median : 24.088   Median :1.0000
##                     Mean   :-31.138   Mean   :  7.792   Mean   :0.5024
```

```
##                    3rd Qu.:  2.938   3rd Qu.: 38.242   3rd Qu.:1.0000
##                    Max.   : 67.241   Max.   : 83.660   Max.   :1.0000
##        ID        BO2_chlomean_ss   BO2_salinitymean_ss   BO_damean
##  Min.   :    1   Min.   :0.01585   Min.   : 0.1632     Min.   :0.02180
##  1st Qu.: 7416   1st Qu.:0.08954   1st Qu.:34.5100     1st Qu.:0.03504
##  Median :14831   Median :0.24526   Median :35.4267     Median :0.05037
##  Mean   :14865   Mean   :0.42527   Mean   :35.2420     Mean   :0.07833
##  3rd Qu.:22313   3rd Qu.:0.68457   3rd Qu.:36.2163     3rd Qu.:0.08800
##  Max.   :29806   Max.   :3.44767   Max.   :39.2467     Max.   :0.69655
##    BO_sstmean
##  Min.   :-1.705
##  1st Qu.:14.853
##  Median :18.899
##  Mean   :18.120
##  3rd Qu.:25.374
##  Max.   :30.663
```

The resulting dataset has 29661. We save this dataset in a local file to work on it in subsequent steps.

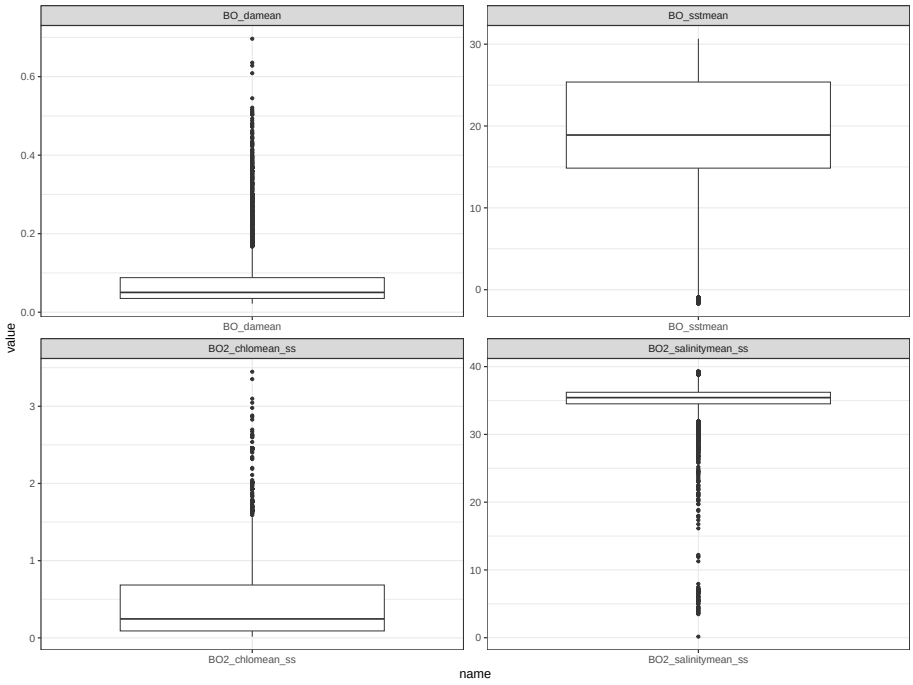
```
save(list="data", file="data/outputs_for_modelling/PAdata_with_env.RData")
```

3.4 Exploratory plots of environmental variables

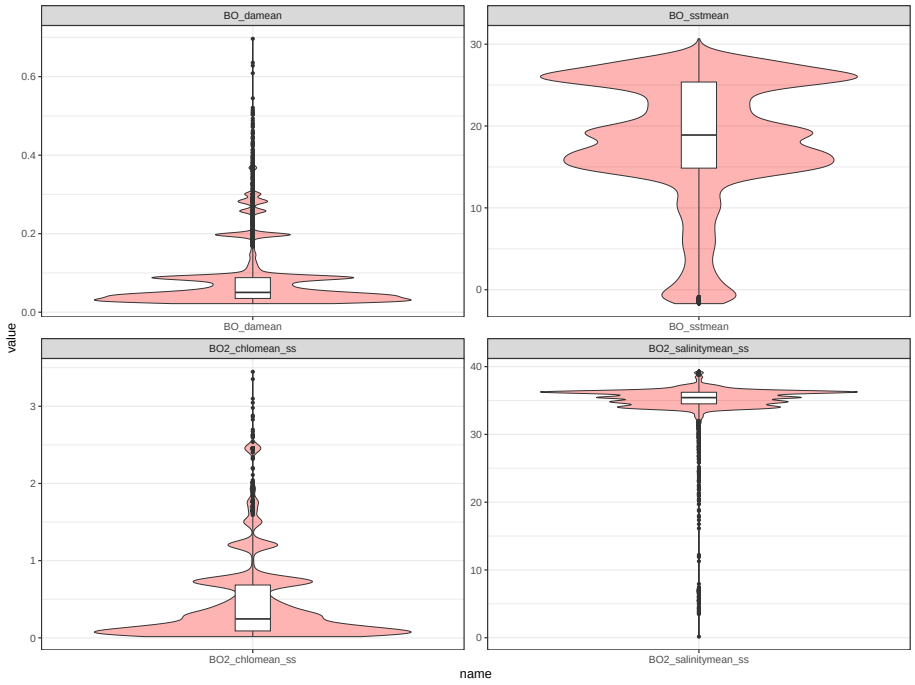
Before starting the modelling process, we are going to explore the individual variables in the dataset.

We can explore the distributions of each of the environmental variables by looking at the violin and boxplots and at the histograms and density plots as follows:

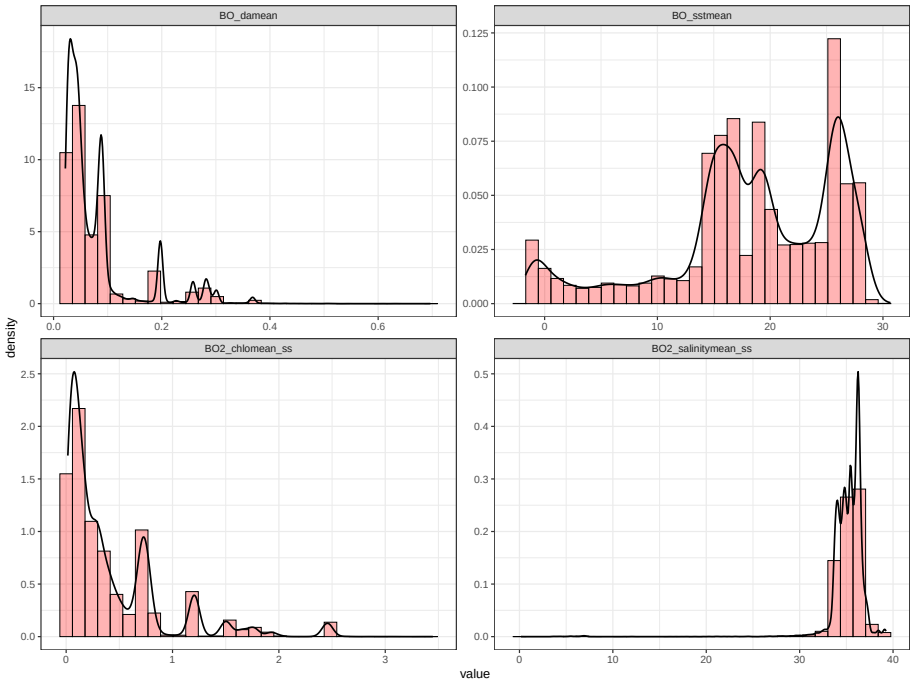
```
tmp <- data[, c("BO2_chlomean_ss","BO2_salinitymean_ss","BO_damean","BO_sstmean")]
tmp <- pivot_longer(data=tmp, cols=everything())

ggplot(data=tmp, aes(x=name, y=value)) +
  geom_boxplot()+
  facet_wrap(~name, scales="free")
```




```
ggplot(data=tmp, aes(x=name, y=value)) +
  geom_violin(fill="red", alpha=0.3)+
  geom_boxplot(width=0.1)+
  facet_wrap(~name, scales="free")
```



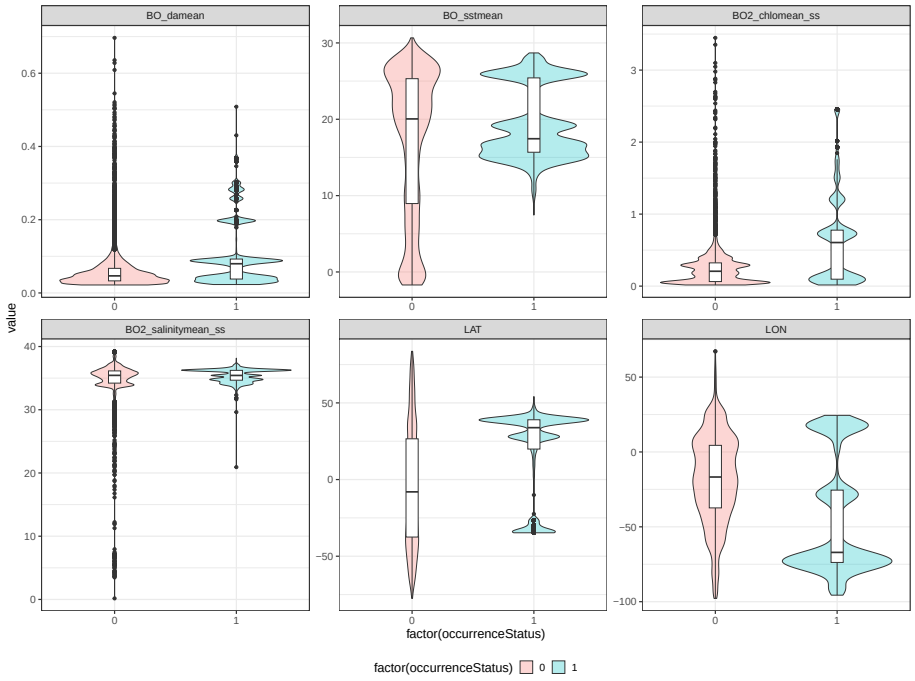

```
ggplot(data=tmp, aes(x=value)) +
  geom_histogram(aes(y= after_stat(density)), colour=1, fill="red", alpha=0.3)+
  geom_density(lwd=1)+
  facet_wrap(~name, scales="free")
```



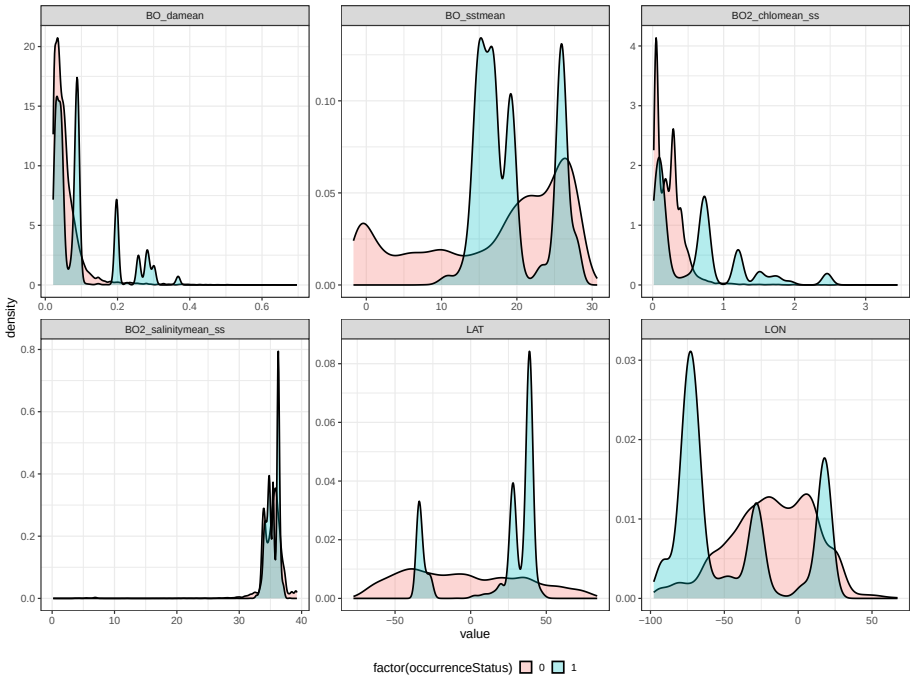
3.5 Exploratory plots of environmental variables depending on presence/absence data

To analyse if there are preferences for certain ranges of the environmental variables, we compare the distribution of the environmental variables for presence and absence data:

```
tmp <- data[, c("LON", "LAT", "BO2_chlomean_ss","BO2_salinitymean_ss","BO_damean","BO_s
tmp <- pivot_longer(data=tmp, cols=!occurrenceStatus)

ggplot(data=tmp, aes(x=factor(occurrenceStatus), y=value, fill=factor(occurrenceStatus)
  geom_violin(alpha=0.3)+
  geom_boxplot(fill="white", width=0.1)+
  facet_wrap(~name, scales="free")+
  theme(legend.position = "bottom",legend.background = element_rect(fill = "white", col
```



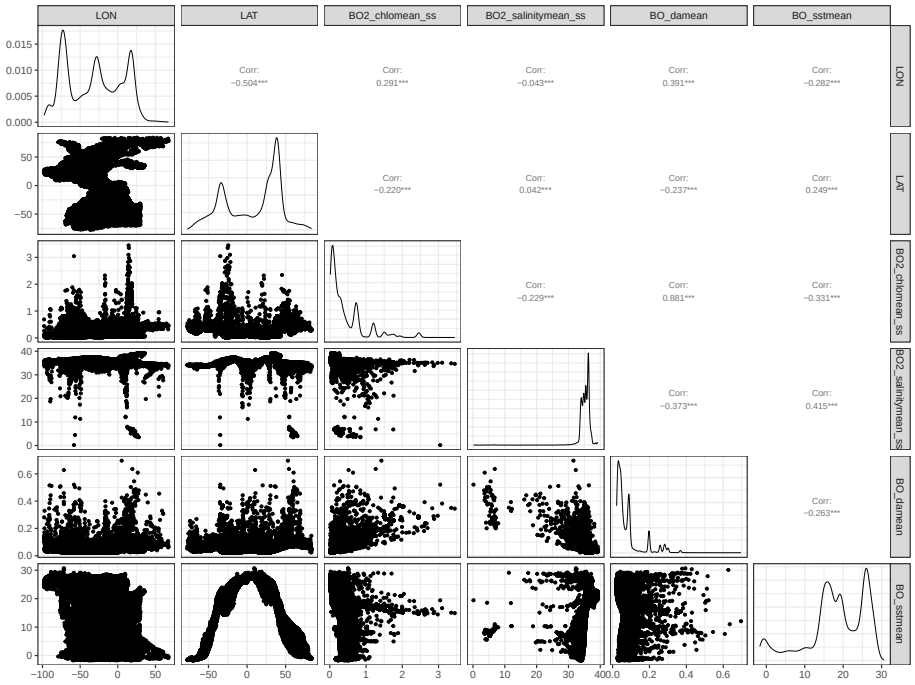

```
ggplot(data=tmp, aes(x=value, fill=factor(occurrenceStatus), group=factor(occurrenceStatus))) +
  geom_density(lwd=1, alpha=0.3)+
  facet_wrap(~name, scales="free")+
  theme(legend.position = "bottom",legend.background = element_rect(fill = "white", colour = NA))
```



3.6 Correlation analysis

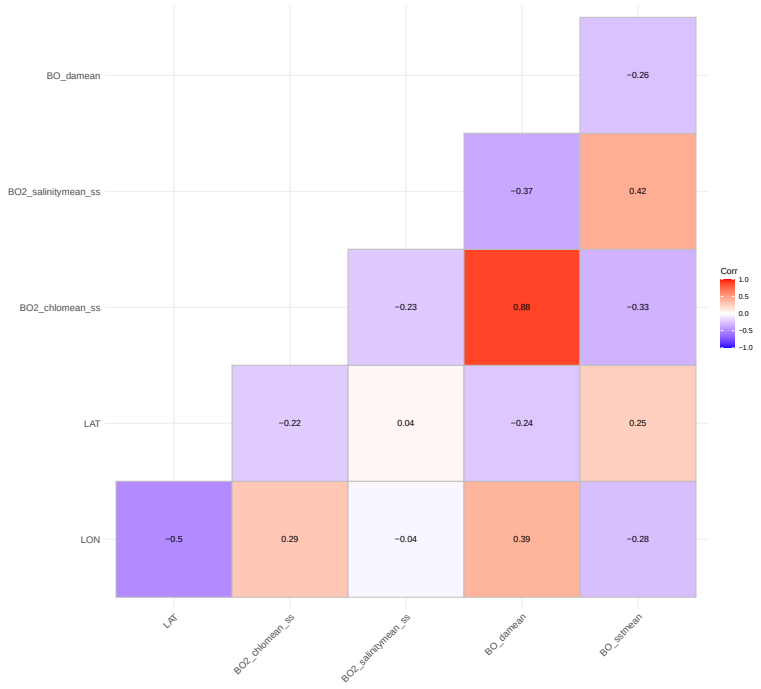
Some of the environmental variables can be correlated. The `GGally` package allows to easily produce pairplots of the variables and their correlation.

```
tmp <- data[, c("LON","LAT","BO2_chlomean_ss","BO2_salinitymean_ss","BO_damean","BO_sst

ggpairs(tmp) #this takes some minutes
```



A more detailed analysis of the potential correlations can be conducted using the package `ggcorrplot`:

```
mat <- cor(tmp, use="complete.obs")
p.mat <- cor_pmat(tmp)

ggcorrplot(mat, type = "lower", lab=T, p.mat = p.mat)
```



3.7 Variance Inflation Factor (VIF)

Furthermore, multicollinearity in regression analysis can be explored using the VIF (Variance Inflation Factor). The value of the VIF statistics indicate the level of multicollinearity with the rest of the variables:

- VIF equal to 1 = variables are not correlated
- VIF between 1 and 5 = variables are moderately correlated
- VIF greater than 5 = variables are highly correlated

There are several packages in `R` that allows to calculate the VIF statistics. In this case we use the package `HH`:

```
# select variables for VIF calculation
v.table <- data %>%
  dplyr::select (BO2_salinitymean_ss, BO_sstmean, BO2_chlomean_ss, BO_damean)

# get VIF results
out.vif <- vif(v.table)
sort(out.vif)
```

```
##          BO_sstmean BO2_salinitymean_ss     BO2_chlomean_ss           BO_damean
##            1.375837            1.493366            5.270639            5.478410
```

We remove the variable that has the highest VIF value and we test again the multicollinearity:

```
v.table <- v.table %>%
  dplyr::select (-BO_damean)

#  get new VIF results
out.vif <- vif(v.table)
sort(out.vif)
```

```
##     BO2_chlomean_ss BO2_salinitymean_ss          BO_sstmean
##            1.136295            1.222233            1.300901
```

Now all the variables have VIF values that are acceptable. So, we proceed to remove BO_damean (Diffuse attenuation coefficient at 490 nm). And save the selected variables for the next modelling stages:

```
data <- data %>% dplyr::select (-BO_damean)
```

We save the dataset as our output for modelling

```
save(list="data", file="data/outputs_for_modelling/PAdata_with_env.RData")
```

3.8 Shape Constrained-Generalized Additive Models

In order to fit SDM in agreement with the ecological niche theory, the proposed Shape-Constrained Generalized Additive Models (SC-GAMs) in (**?**) are fitted in this section. SC-GAMs are based on Generalized Additive Models, allowing us to impose shape-constraints to the linear predictor function. The R package SCAM implements the general framework developed by (**?**) using shape-constrained P-splines. Monotonicity and concavity/convexity constraints can be imposed on the sign of the first and/or the second derivatives of the smooth terms. For fitting Species Distribution Models in agreement with the ecological niche theory, we imposed concavity constraints ($f''(x) \leq 0$), so that the response can presents at most a single mode.

Alternatively, the R package `mboost` fits SC-GAMs using boosting methods. We are not going to develop this alternative here.

First we load the list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow
  "stringr",
  "RColorBrewer",
  "ggplot2",
  "dplyr",

  #SPATIAL DATA --------#
  "rgdal",
  "fields",
  "maps" ,
  "raster",

  #MODEL FIT --------#
  "scam",
  "plotmo",
  "SDMTools",
  "pkgbuild",
  "dismo"
  )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

install_load_function(requiredPackages)
```

```
##         here   rstudioapi      stringr RColorBrewer      ggplot2        dplyr
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##        rgdal       fields         maps       raster         scam       plotmo
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##     SDMTools     pkgbuild        dismo
##         TRUE         TRUE         TRUE
```

Install and load SDMTools. You need to manually install RTools installed: https://cran.r-project.org/bin/windows/Rtools/history.html

```
# find_rtools()
#
# install.packages("remotes")
# remotes::install_version("SDMTools", version = "1.1-221.2")
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

3.9 Model fit

We set the working directory to the folder where the current script is located and we load the dataset (PAdata_with_env.Rdata) containing the presence-absence data together with the environmental data.

```
load(here::here ("data", "outputs_for_modelling", "PAdata_with_env.Rdata"))
```

To fit a logistic regression model in the SC-GAMs framework, we use the scam function, where we set the binomial family with the logit link function. Our response variable is the presence-absence data and the selected three explanatory variables are the SST, chlorophyll and salinity. Each variable is included in the model through an spline function where the concavity constraint is set using bs="cv". The details about this option can be found in the section "Constructor for concave P-splines in SCAMs" of the SCAM manual (https://cran.r-project.org/web/packages/scam/scam.pdf). The number of knots (k) is fixed at 8 in this example for a good balance between flexibility and computation time.

UNIVARIATE MODELS

Before fitting the model with the selected three environmental variables, we can fit univariate model as follows.

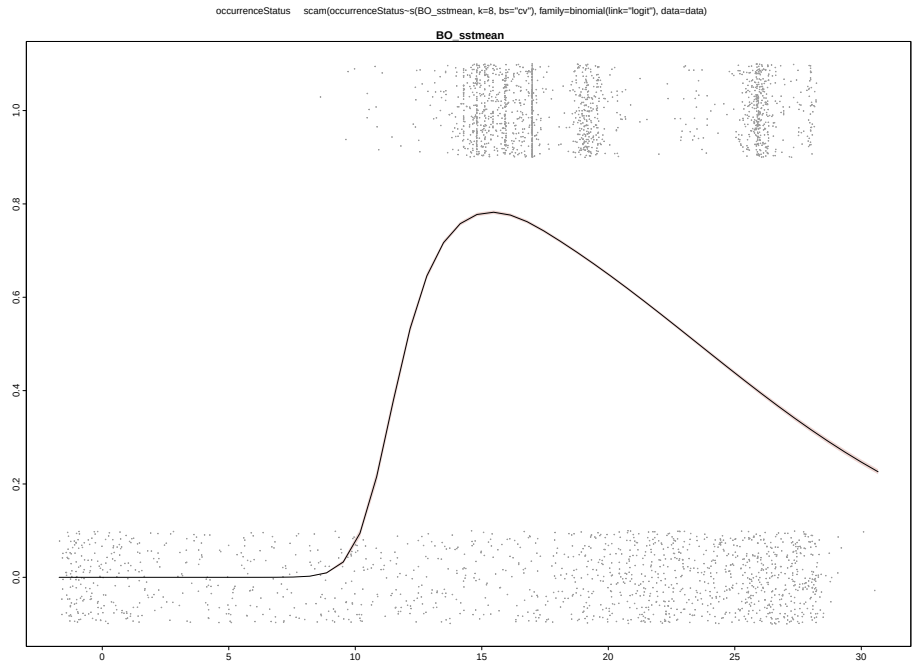
We fit the univariate model for SST, we print the summary of the model fit, and look at the fitted curve in the response scale.

```
model_sst <- scam (occurrenceStatus ~  s(BO_sstmean, k=8,bs="cv"), family=binomial(link="logit"),
summary(model_sst)
```

```
##
## Family: binomial
## Link function: logit
##
```

```
## Formula:
## occurrenceStatus ~ s(BO_sstmean, k = 8, bs = "cv")
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -47.180      2.182  -21.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df Chi.sq p-value
## s(BO_sstmean)   2      2   1111  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.2607   Deviance explained = 23.1%
## UBRE score = 0.066363  Scale est. = 1         n = 29661
```

```
plotmo(model_sst,level = 0.95, pt.col=8)
```

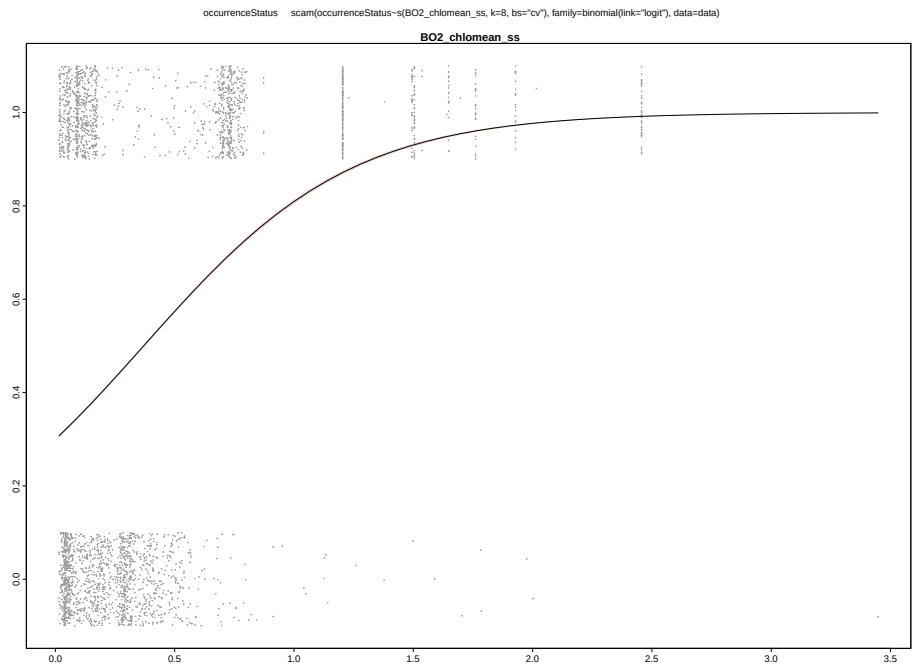


We repeat the same steps for the rest of the variables.

```
model_chl <- scam (occurrenceStatus ~  s(BO2_chlomean_ss, k=8,bs="cv"), family=binomial(link="log
summary(model_chl)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ s(BO2_chlomean_ss, k = 8, bs = "cv")
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.3878     0.0417  -57.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df Chi.sq p-value
## s(BO2_chlomean_ss)   1      1   3263  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Rank: 7/8
##
## R-sq.(adj) =  0.1546   Deviance explained = 12.4%
## UBRE score = 0.21423  Scale est. = 1         n = 29661
```

```
plotmo(model_chl,level = 0.95, pt.col=8)
```



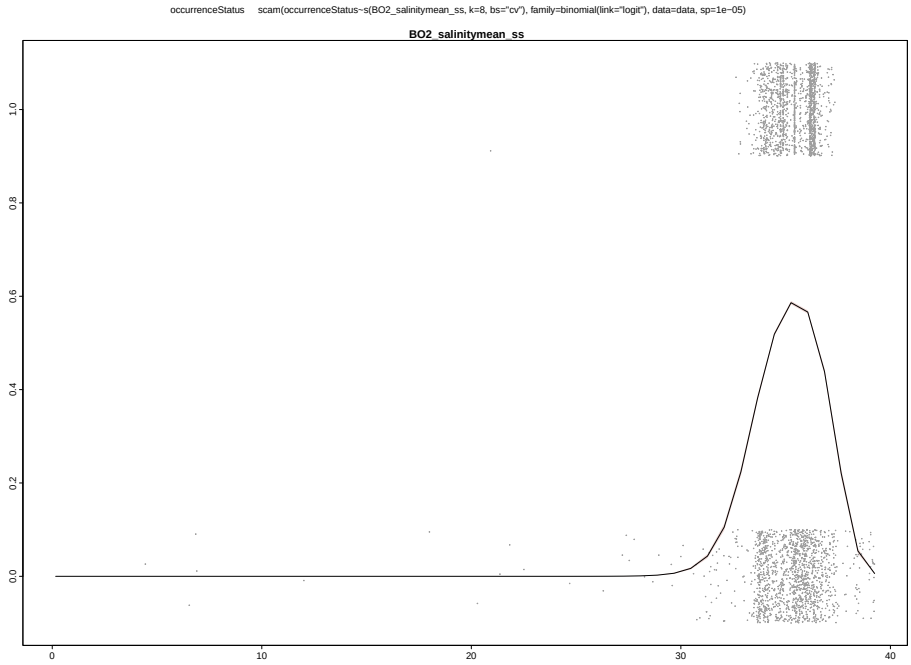
Due to convergence issues, sometimes it is necessary to fix the smoothing parameter (sp) at small value, i.e. 10^{-5}, as here when introducing salinity as an explanatory variable. If no value is provided, the smoothing parameter is estimated within the model.

```
model_sal <- scam (occurrenceStatus ~  s(BO2_salinitymean_ss, k=8,bs="cv"), family=bino
summary(model_sal)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ s(BO2_salinitymean_ss, k = 8, bs = "cv")
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -50.076      1.853  -27.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                         edf Ref.df Chi.sq p-value
```

```
## s(BO2_salinitymean_ss)   2      2  913.1  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0514   Deviance explained = 4.38%
##   Scale est. = 1         n = 29661
```

```
plotmo(model_sal,level = 0.95, pt.col=8)
```



MODEL WITH ALL VARIABLES

Now we fit the model including the three selected variables.

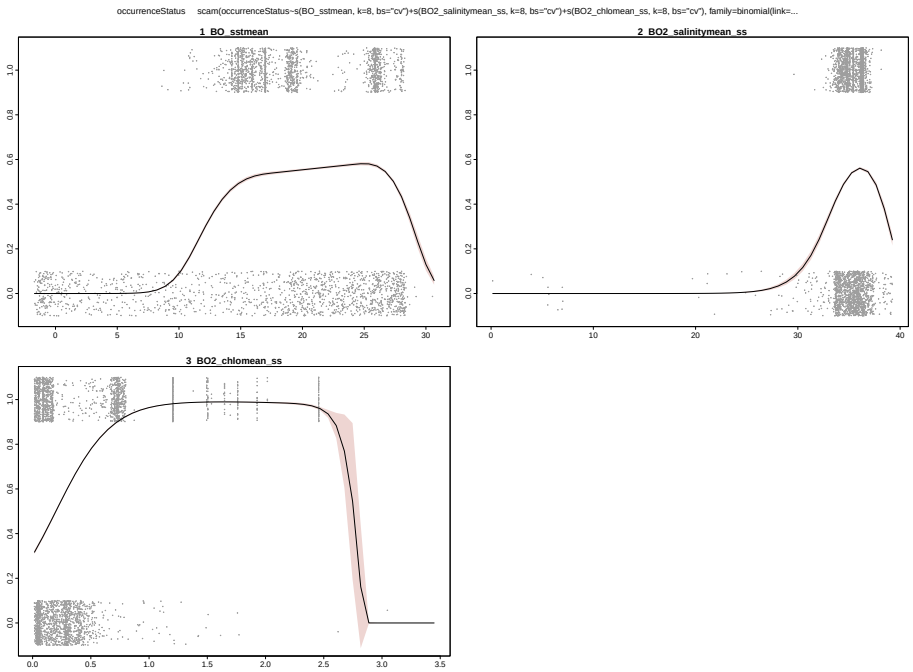
```
model <- scam (occurrenceStatus ~  s(BO_sstmean, k=8,bs="cv")+ s(BO2_salinitymean_ss, k=8,bs="cv'
summary(model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ s(BO_sstmean, k = 8, bs = "cv") + s(BO2_salinitymean_ss,
##     k = 8, bs = "cv") + s(BO2_chlomean_ss, k = 8, bs = "cv")
```

```
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -56.08      45.69  -1.227     0.22
##
## Approximate significance of smooth terms:
##                           edf Ref.df Chi.sq p-value
## s(BO_sstmean)           3.936  3.999  772.6  <2e-16 ***
## s(BO2_salinitymean_ss) 2.001  2.001  191.4  <2e-16 ***
## s(BO2_chlomean_ss)     3.985  4.003 1820.7  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.3488   Deviance explained = 31.9%
##   Scale est. = 1         n = 29661
```

```
plotmo(model,level = 0.95, pt.col=8)
```

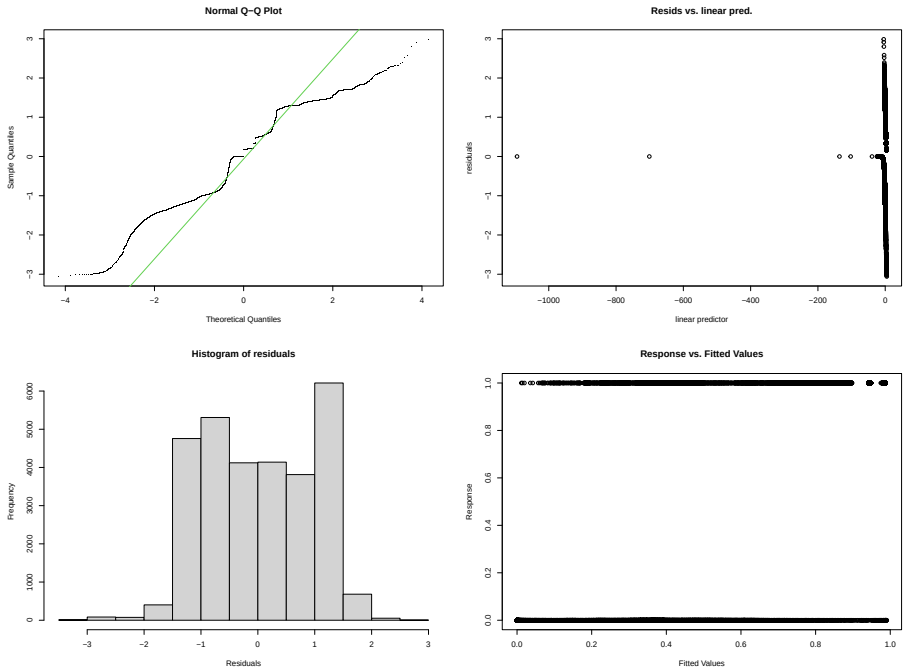
```
##  plotmo grid:    BO_sstmean BO2_salinitymean_ss BO2_chlomean_ss
##                    18.89921             35.42666       0.2452632
```



We can see in the summary of the fit, that all included variables are statistically significant (with $p<0.05$) and present a unimodal response curve.

For a more detailed check of the fitting, we can use the scam.check function:

```
scam.check(model)
```




```
##
## Method: UBRE   Optimizer: NA newton
## The optimal smoothing parameter(s): 1e-05 1e-05 1e-05 .
```

3.10 Model selection

When several explanatory variables are available, a variable selection process can be carried out. Here we provide as an example, a function that performs forward variable selection (modsel.scam) based on the significance of variables and AIC values of the fits.

```
source("function/function_scam_selection_optimized.R")
```

We save the names of the variables we want to introduce for the variable selection process as a vector:

```
vars <- c("BO_sstmean",
          "BO2_salinitymean_ss",
          "BO2_chlomean_ss")
```

The default AIC tolerance is 2 and there is not a limit on selected terms in this example. These options can be modified through aic.tol and vmax arguments in the function. The number of knots and the sp can be also modified.

```
model_SCGAM <- try(modsel.scam(basef="occurrenceStatus ~ 1", vars=vars, dat=data,sp=0.
```

```
## [1] "Fitting models with 1 terms"
##   occurrenceStatus ~ 1+s(BO_sstmean,m=2,bs='cv',k=8)
##   occurrenceStatus ~ 1+s(BO2_salinitymean_ss,m=2,bs='cv',k=8)
##   occurrenceStatus ~ 1+s(BO2_chlomean_ss,m=2,bs='cv',k=8)
## [1] "Fitting models with 2 terms"
##   occurrenceStatus ~ 1+s(BO_sstmean,m=2,bs='cv',k=8)+s(BO2_salinitymean_ss,m=2,bs='
##   occurrenceStatus ~ 1+s(BO_sstmean,m=2,bs='cv',k=8)+s(BO2_chlomean_ss,m=2,bs='cv',
## [1] "Fitting models with 3 terms"
##   occurrenceStatus ~ 1+s(BO_sstmean,m=2,bs='cv',k=8)+s(BO2_chlomean_ss,m=2,bs='cv',
```

We check results of the selected model, such as, selected variable names:

```
model_SCGAM$svars
```

```
## [1] "BO_sstmean"          "BO2_chlomean_ss"     "BO2_salinitymean_ss"
```

AICs of the fitted models:

```
sapply(model_SCGAM$smod, AIC)
```

```
##                null          BO_sstmean     BO2_chlomean_ss BO2_salinitymean_ss
##            41120.17            31629.43            28388.33            28032.06
```

Explained deviance of fitted models:

```
sapply(model_SCGAM$smod, function(x) summary(x)$dev.expl)
```

```
##                null          BO_sstmean     BO2_chlomean_ss BO2_salinitymean_ss
##       3.539048e-16        2.309134e-01        3.100285e-01        3.187874e-01
```

Formulas of the fitted models:

```
lapply(model_SCGAM$smod, formula)
```

```
## $null
## occurrenceStatus ~ 1
## <environment: 0x000002667cc23298>
##
## $BO_sstmean
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
##
## $BO2_chlomean_ss
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8) +
##     s(BO2_chlomean_ss, m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
##
## $BO2_salinitymean_ss
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8) +
##     s(BO2_chlomean_ss, m = 2, bs = "cv", k = 8) + s(BO2_salinitymean_ss,
##     m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
```

Summaries of the fitted models:

```
lapply(model_SCGAM$smod, summary)
```

```
## $null
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ 1
## <environment: 0x000002667cc23298>
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.009777   0.011613   0.842      0.4
##
##
## R-sq.(adj) =      0   Deviance explained = 3.54e-14%
##   Scale est. = 1         n = 29661
##
##
## $BO_sstmean
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -47.30       3.12  -15.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df Chi.sq p-value
## s(BO_sstmean)   2      2  987.3  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.2607   Deviance explained = 23.1%
##   Scale est. = 1         n = 29661
##
##
## $BO2_chlomean_ss
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8) +
##     s(BO2_chlomean_ss, m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -40.38      89.57  -0.451    0.652
##
## Approximate significance of smooth terms:
##                       edf Ref.df Chi.sq p-value
## s(BO_sstmean)       3.978  4.004  938.3  <2e-16 ***
## s(BO2_chlomean_ss) 4.002  4.005 2374.3  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.3399   Deviance explained =   31%
```

```
##   Scale est. = 1         n = 29661
##
##
## $BO2_salinitymean_ss
##
## Family: binomial
## Link function: logit
##
## Formula:
## occurrenceStatus ~ 1 + s(BO_sstmean, m = 2, bs = "cv", k = 8) +
##     s(BO2_chlomean_ss, m = 2, bs = "cv", k = 8) + s(BO2_salinitymean_ss,
##     m = 2, bs = "cv", k = 8)
## <environment: 0x000002667cc23298>
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -56.08      45.69  -1.227     0.22
##
## Approximate significance of smooth terms:
##                           edf Ref.df Chi.sq p-value
## s(BO_sstmean)           3.936  3.999  772.6  <2e-16 ***
## s(BO2_chlomean_ss)      3.985  4.003 1820.7  <2e-16 ***
## s(BO2_salinitymean_ss) 2.001  2.001  191.4  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.3488   Deviance explained = 31.9%
##   Scale est. = 1         n = 29661
```

The last model of the list, is the selected one. Which in this case contains the considered three variables.

```
selected_model <- model_SCGAM$smod[[length(model_SCGAM$smod)]]
```

We save the selected model

```
save(list="selected_model", file="models/selected_model.RData")
```

Note that there are multiple options and criteria for model selection that are not reviewed here. Any model selection technique used for GAMs can be used also for SC-GAMs.

Chapter 4

Model validation

In this section, model validation is performed in order to assess the predictive performance of the selected model. This validation is conducted via k-fold cross-validation. The data set is divided into k equally sized groups (**?**), using a percentage of randomly selected observations to run the model and the remaining for validation, iteratively for each fold.

First we load the list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow
  "stringr",
  "RColorBrewer",
  "ggplot2",
  "dplyr",
  "tidyverse",
  "R.utils",
  "ggpubr",
  "hrbrthemes",

  #SPATIAL DATA --------#
  "rgdal",
  "fields",
  "maps" ,
  "raster",

  #MODEL FIT --------#
  "scam",
  "plotmo",
```

```
  "SDMTools",
  "pkgbuild",
  "dismo"
  )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

install_load_function(requiredPackages)
```

```
##         here   rstudioapi      stringr RColorBrewer      ggplot2        dplyr
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##    tidyverse      R.utils       ggpubr   hrbrthemes        rgdal       fields
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##         maps       raster         scam       plotmo     SDMTools     pkgbuild
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##        dismo
##         TRUE
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

We load output from the selected model saved in the previous step.

```
load(here::here ("models", "selected_model.Rdata"))
```

4.1 Optimum threshold

We generate a data frame with the data used in the selected model and we add the predicted values.

```
  #PAdata_enviroment used in the selected model
  data<-selected_model$model

  # Predict
  scgam.pred <- predict(selected_model, newdata=data, type="response")

  # Add the prediction to the data object
  data$scgam.pred <- as.vector(scgam.pred)
  head(data)
```

```
##    occurrenceStatus BO_sstmean BO2_chlomean_ss BO2_salinitymean_ss scgam.pred
## 1                 1   16.99439      1.20410309            35.42666  0.9804524
## 5                 1   16.99439      1.20410309            35.42666  0.9804524
## 8                 1   25.52963      0.08455528            36.43070  0.4238477
## 10                1   16.99439      1.20410309            35.42666  0.9804524
## 11                1   15.21486      0.72283228            34.20426  0.8449155
## 12                1   16.99439      1.20410309            35.42666  0.9804524
```

The threshold for presence-absence classification for each species is obtained as the values maximizing sensitivity plus specificity (**?**). If the result was a range (instead of a single value), we would select the mean value of the range.

```
  # Optimizing the threshold probability
  obs <- data$occurrenceStatus
  predSCGAM_P <- data$scgam.pred

  # threshold optimizing
  myoptim <- optim.thresh (obs,predSCGAM_P)
  myoptim
```

```
## $min.occurence.prediction
## [1] 0.01156124
##
## $mean.occurence.prediction
## [1] 0.6642745
##
## $`10.percent.omission`
## [1] 0.36
##
## $`sensitivity=specificity`
## [1] 0.44
##
## $`max.sensitivity+specificity`
## [1] 0.42
```

```
##
## $maxKappa
## [1] 0.42
##
## $max.prop.correct
## [1] 0.42
##
## $min.ROC.plot.distance
## [1] 0.42
```

```
  # select the threshold that maximizes the sum of sensitivity and specificity
  myThreshold <- as.numeric((myoptim[["max.sensitivity+specificity"]]))
```

Accuracy indicators, such as AUC (Area Under the Receiver Operating Characteristic—ROC—curve), sensitivity (true predicted presences) and specificity (true predicted absences) are first computed for the all observations.

```
  # Accuracy values with all observations
  accuracy (obs, predSCGAM_P, threshold=myThreshold)
```

```
##   threshold       AUC omission.rate sensitivity specificity prop.correct
## 1      0.42 0.7415229     0.1985506   0.8014494   0.6815964    0.7418159
##      Kappa
## 1 0.483323
```

```
  # Create confusion matrix with all observations
  confusion.matrix(obs, predSCGAM_P, threshold=myThreshold)
```

```
##     obs
## pred     0     1
##    0 10059  2959
##    1  4699 11944
## attr(,"class")
## [1] "confusion.matrix"
```

4.2 k-fold validation

In this case we use a 5-fold cross-validation.

```
  # Number of groups
  k <- 5
  # Generate groups
  groups<-kfold(data, k, by=data$occurrencestatus)
```

The model is run for each of the 5 random subset (with a 20% of the observations) and indicators are then computed using the remaining 80% of the observations. Indicators are the averaged across folds.

```
  # Initialise the confusion matrix and the accuracy table:
  myCM <- NULL
  myACC <- NULL

  # get the formula of the selected model
  formula <- summary(selected_model)[["formula"]]

  # get the smoothing parameters of the selected model
  sp <- selected_model$sp

  # loop for each group k
  for (j in 1:k) {
    # Preparation of Training Sites
    p_Training <- data[groups != j,]

    # Model fit
    selected_model.sp.j <- scam (formula, family=binomial(link="logit"),data=p_Training, sp=c(sp)

    # Predict Model
    p_validacion<-data[groups == j,]

    selected_model.sp.j.pred <- predict(selected_model.sp.j, newdata=p_validacion, type="response
    p_validacion$Pred <- selected_model.sp.j.pred

    # Confussion matrix and accuracy table for fold j
    obs <- p_validacion$occurrenceStatus
    predSCGAM <- p_validacion$Pred
    myCM <- rbind(myCM, as.numeric(confusion.matrix(obs, predSCGAM, threshold=myThreshold)))
    myACC <- rbind(myACC, accuracy(obs, predSCGAM, threshold=myThreshold))
  }

  # Mean values across k-folds
  validation_summary<-cbind(Threshold=myThreshold,
                            mean_AUC=mean(myACC$AUC),
                            mean_Omision=mean(myACC$omission.rate),
                            mean_sensitivity=mean(myACC$sensitivity),
                            mean_specificity=mean(myACC$specificity),
                            mean_Prop.Corr=mean(myACC$prop.correct))

  validation_summary
```

```
##      Threshold mean_AUC mean_Omision mean_sensitivity mean_specificity
```

```
## [1,]      0.42 0.725464    0.2399716        0.7600284        0.6908995
##      mean_Prop.Corr
## [1,]      0.7256674
```

We save the validation summary object.

```
save(validation_summary, file = here::here("models/validation_summary.RData"))
```

Chapter 5

Prediction and maps

In this chapter we predict from the fitted model and produce final SDMs maps.

First we load a list of required libraries.

```
requiredPackages <- c(
  #GENERAL USE LIBRARIES --------#
  "here", # Library for reproducible workflow
  "rstudioapi",  # Library for reproducible workflow
  "ggplot2", #for plotting
  "tidyverse",
  "rgdal", # to work with Spatial data
  "raster", #spatial
  "maps", #world map
  "maptools", #plotting world map
  "RColorBrewer", #color palette
  "scam", #sdm models under the ecological niche theory framework
  "ggpubr"
  )
```

We run a function to install the required packages that are not in our system and load all the required packages.

```
install_load_function <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

install_load_function(requiredPackages)
```

```
##         here   rstudioapi      ggplot2    tidyverse        rgdal       raster
##         TRUE         TRUE         TRUE         TRUE         TRUE         TRUE
##         maps     maptools RColorBrewer         scam       ggpubr
##         TRUE         TRUE         TRUE         TRUE         TRUE
```

We define some overall settings.

```
# General settings for ggplot (black-white background, larger base_size)
theme_set(theme_bw(base_size = 16))
```

5.1 Prepare environmental data.

In previous steps (see Chapter 2), we have defined the study area that defines the extent of our spatial data. We load the `study_area` object that is a SpatialPolygonsDataFrame class:

```
load(here::here ("data", "spatial", "study_area.RData"))
```

And we load the rasterStack with the downloaded environmental data.

```
mylayers<-stack("data/env/mylayers.tif")
```

We transform the environmental data set first into a data frame, and then into a SpatialDataFrame.

```
env_dataframe <- raster::as.data.frame(mylayers, xy=TRUE)

summary(env_dataframe)
```

```
##        x                 y             mylayers_1        mylayers_2
##  Min.   :-97.79   Min.   :-82.96   Min.   :0.0       Min.   : 0.1
##  1st Qu.:-56.23   1st Qu.:-39.73   1st Qu.:0.1       1st Qu.:33.8
##  Median :-14.67   Median :  3.50   Median :0.3       Median :34.6
##  Mean   :-14.67   Mean   :  3.50   Mean   :0.3       Mean   :34.4
##  3rd Qu.: 26.90   3rd Qu.: 46.73   3rd Qu.:0.4       3rd Qu.:35.6
##  Max.   : 68.46   Max.   : 89.96   Max.   :3.6       Max.   :40.7
##                                    NA's   :1501044   NA's   :1501044
##    mylayers_3        mylayers_4
##  Min.   :0.0       Min.   :-1.8
##  1st Qu.:0.0       1st Qu.: 1.9
##  Median :0.0       Median :15.1
##  Mean   :0.1       Mean   :13.7
```

```
##  3rd Qu.:0.1       3rd Qu.:24.1
##  Max.   :1.0       Max.   :32.3
##  NA's   :1652879   NA's   :1652879
```

```
names(env_dataframe) <- c("x", "y", "BO2_chlomean_ss", "BO2_salinitymean_ss", "BO_damean", "BO_ss
```

5.2 Projection

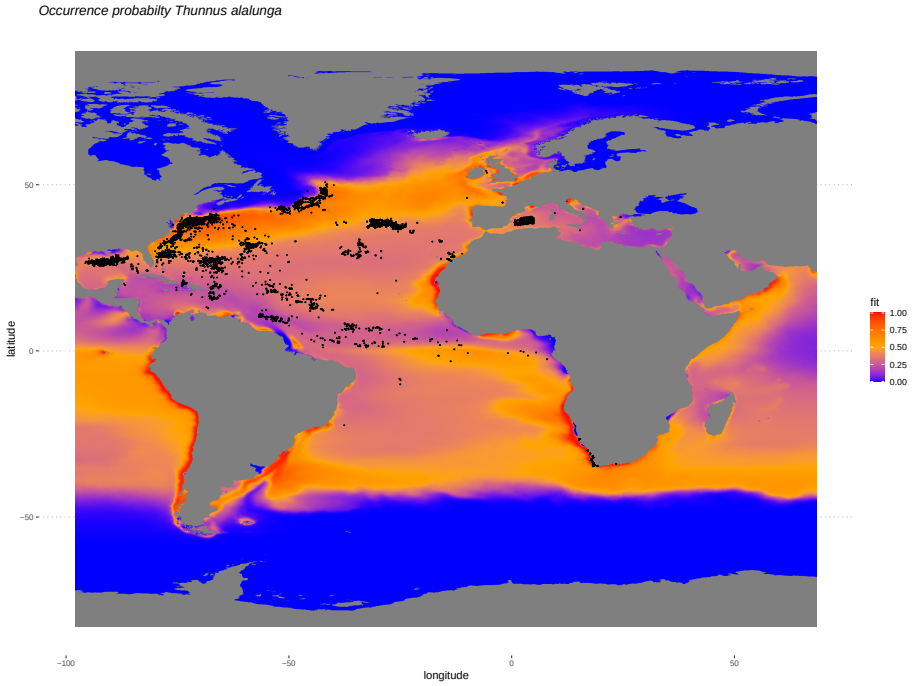
We load the selected model and predict into the whole environmental data.

```
#Load SC-GAM model
load(here::here("models", "selected_model.Rdata"))

# predicting
predict <- predict(selected_model,newdata=env_dataframe,type ="response",se.fit=T)

env_dataframe$fit<-predict$fit
env_dataframe$se.fit<-predict$se.fit

save(env_dataframe, file="results/projection.Rdata")
```

5.3 Mapping

```
#load PA data
load(here::here ("data", "outputs_for_modelling", "PAdata_with_env.Rdata"))


proj_map <-ggplot()+
  geom_raster(data=subset(env_dataframe),
              aes(x,y,fill=fit)) +
  scale_fill_gradient2(low="blue",
                       mid="orange",
                       high="red",
                       midpoint = 0.5,
                       limits = c(0,1)) +
  ggtitle("Occurrence probabilty Thunnus alalunga")+
  geom_point(data=subset(data,occurrenceStatus==1),
             aes(LON,LAT),
             col=1,
             size=0.3) +
  theme_pubclean(base_size = 14)+
```

```
  theme(panel.background = element_blank(),
        plot.title = element_text(face = "italic"),
        #text = element_text(size = 14),
        axis.text.x = element_text(size = 10),
        axis.text.y = element_text(size = 10),
        legend.position="right") +
  labs(y="latitude", x = "longitude")

print(proj_map)
```



We finally save the projection map.

```
ggsave(filename= "Thunnus_alalunga_proj_map.tif",
        plot=proj_map,
        device="tiff",
        path=here::here ("plots", "projections"),
        height=22, width=30,
        units="cm", dpi=300)
```