# Mapinguari

*Gabriel Caetano, Juan Santos, Barry Sinervo*

*2019-09-10*

## Contents



## Introduction

Species distribution models or SDMs are popular tools for predicting individual species distributions and project the effects of climate change on those distributions, under different scenarios. Most SDMs are correlative and do not take into account biological processes underlying species responses to environmental variables. We aim to provide a modeling tool to help fill this gap by estimating geographical layers with biologically relevant information that can be used in SDMs or other biogeographical analysis.

Mapinguari is a package for program R aimed at providing tools to facilitate the incorporation of biological processes in biogeographical analyses. It offers conveniences in fitting, comparing and extrapolating models of biological processes such as physiology and phenology. These spatial extrapolations can be informative by themselves, but also complement traditional correlative SDM methods, by mixing environmental and process-based predictors.

On this manual I will provide some examples of the kind of information that can be generated with Mapinguari, using a terrestrial ectotherm and processes relevant to those animals as examples, such as temperature limited activity times and locomotor performance. The spatial information that can be generated with the package is not limited to those presented here, and the same principles apply to generating surfaces relevant to other taxa, such as metabolic rates, time inside thermal neutral zone, photosyntesis, seed and egg development rates, etc.

# Basics

## Installation

The package is currently hosted on GitHub, this is how to install it:

```
# You need package devtools to install packages from GitHub
library(devtools)

install_github("gabrielhoc/Mapinguari")
```

Next, we are going to give an example of how to use the package, using an example daraset:

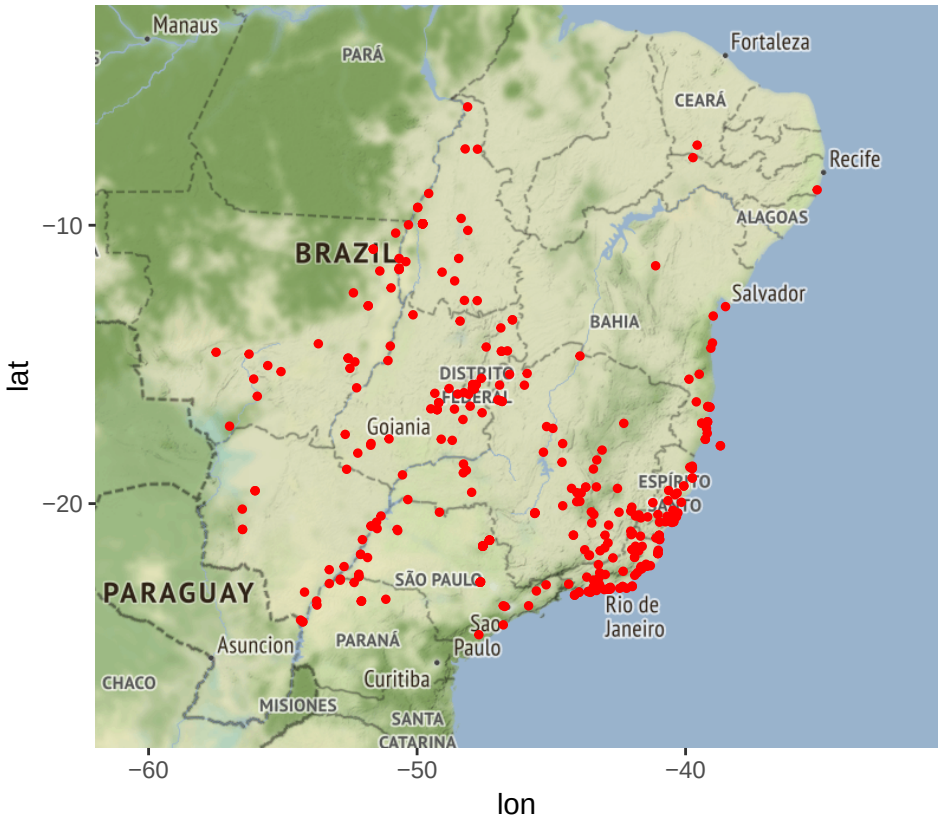
## Example dataset

*Tropidurus torquatus* is a species of lizard from South America. We are going to project daily time of activity and locomotor performance estimates for this species across its distribution and use that to estimate the species distribution. First let's look at the points where it is currently known to occurr. To access the table with distribution data `TtorquatusDistribution`, package Mapinguari has to be loaded. Here we use ggmap to plot the points over an image from Open Street Maps.

```
library(Mapinguari)
library(magrittr)

# First, let's take a look at TtorquatusDistribution

head(TtorquatusDistribution)
```

```
##                species       Lon       Lat
## 1 Tropidurus_torquatus -39.19280 -17.51809
## 2 Tropidurus_torquatus -39.49000 -15.41000
## 3 Tropidurus_torquatus -39.26521 -17.73436
## 4 Tropidurus_torquatus -39.26521 -17.73436
## 5 Tropidurus_torquatus -38.70277 -17.96281
## 6 Tropidurus_torquatus -38.70277 -17.96281
```

```
library(ggmap)
library(ggplot2)

Ttorquatus_bbox <- make_bbox(lat = Lat,
                             lon = Lon,
                             data = TtorquatusDistribution,
                             f = 0.2)

Ttorquatus_big <- get_map(location = Ttorquatus_bbox,
                          source = "osm",
                          maptype = "terrain")

ggmap(Ttorquatus_big) +
  geom_point(data = TtorquatusDistribution,
             mapping = aes(x = Lon, y = Lat),
             size = 1,
             colour = 'red')
```




## Cleaning points

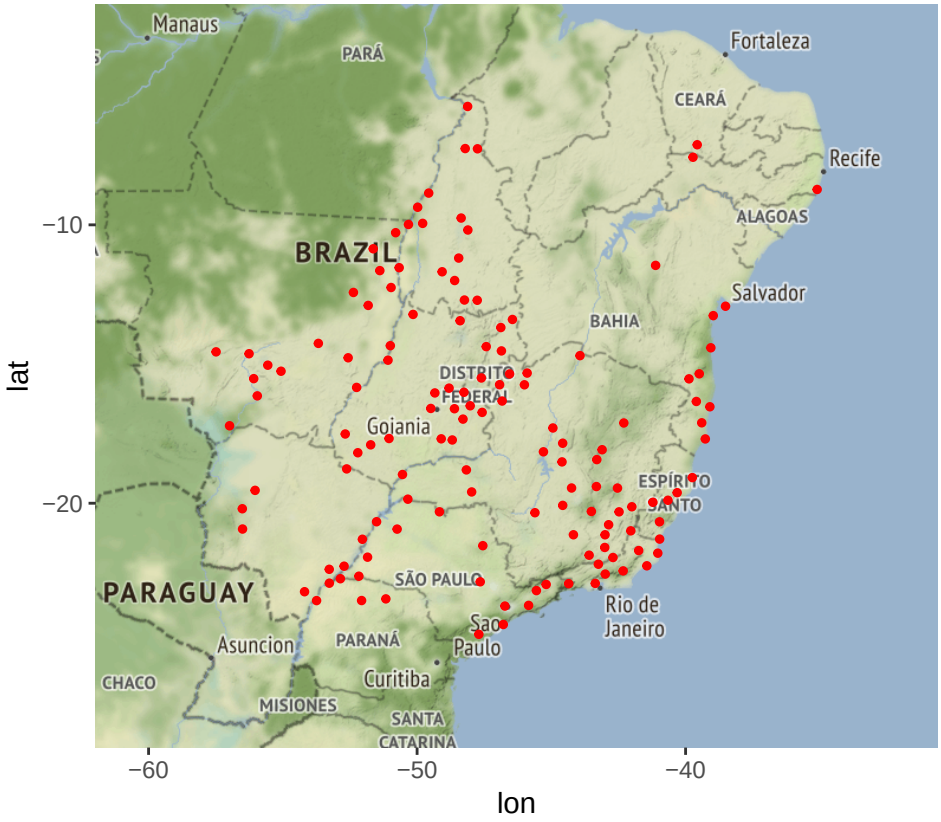
Some areas on the map, such as the southwest of Brazil around the states of Rio de Janeiro and Espírito Santo seem to have much more points than others. That is likely due to biases in sampling, which we do not want to introduce in any spatial analysis. Other points might be mistakenly placed in regions where the species is known not to occurr, such as the ocean. Let's use function `clean_points` to filter out points that are within 2 kilometers from each other, in order to rarefy the points and reduce sampling bias, and removeany point that might fall in the ocean. We can use an altitude layer to identify the ocean areas.

```
# First, we need to obtain an altitude raster to filter by altitude.
library(raster)
alt <- raster::getData("alt", country = "BRA", mask = TRUE)

# Then, we clean the points
TtorquatusDistribution_clean <-
  clean_points(coord = TtorquatusDistribution,
               merge_dist = 40000, # eliminate points within 40 km from each other
               filter_layer = !is.na(alt)) # keeps poins only where alt is not NA
```

```
##                   [,1]
## n_entries_species  359
## n_entries_clean    139
```

In this case, we eliminated points closer to 40 km from each other and those that fall in a location with value NA in our altitude raster (ocean).

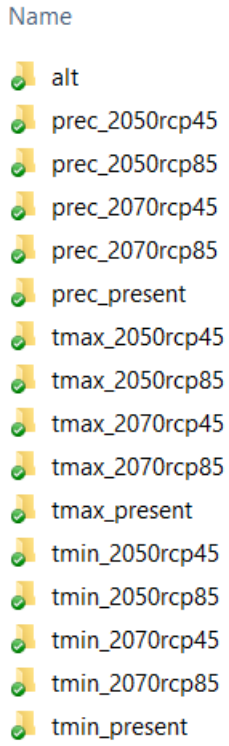
```
ggmap(Ttorquatus_big) +
  geom_point(data = TtorquatusDistribution_clean,
             mapping = aes(x = Lon, y = Lat),
             size = 1,
             colour = 'red')
```




Out of 359 points, we are left with only 145. For now we will use these points only to delimitate our study area, but they will be more useful in later chapters. Now, let's get some information on the climatic for this area.

## Obtaining environmental rasters

The WorldClim database (Hijmans et al, 2005, available at: https://www.worldclim.org) provides current climatic data for the whole world, as well as projections for the past and future, modeled under different scenarios of carbon emission (representative concentration pathways - RCPs). I have downloaded some of those files in 10 minutes resolution and placed them in my global_grids_10min folder. I'm using 10 minutes resolution so the examples run faster, but you should consider which resolution is more appropriate for your kind of analysis. Here is how it looks inside the folder:




Notice that I created sub folders for every combination of variable and scenario. Inside those sub folders are the corresponding raster files downloaded from WorldClim or otherwise, containing the geographical information on the variable for that scenario. You will notice I used a convention *variable_scenario* when naming the folders. This might seem like a hassle now, but it will save time later, as this structure will allow us to easily choose, load, organize and crop those rasters in one step, using function `get_rasters`, detailed on the next section.

Let's assign the path to the new directory to an object, so we don't have to type it every time:

```
mydir <- '/Volumes/Podocnemis/Gabriel/Dropbox/Data/'
```

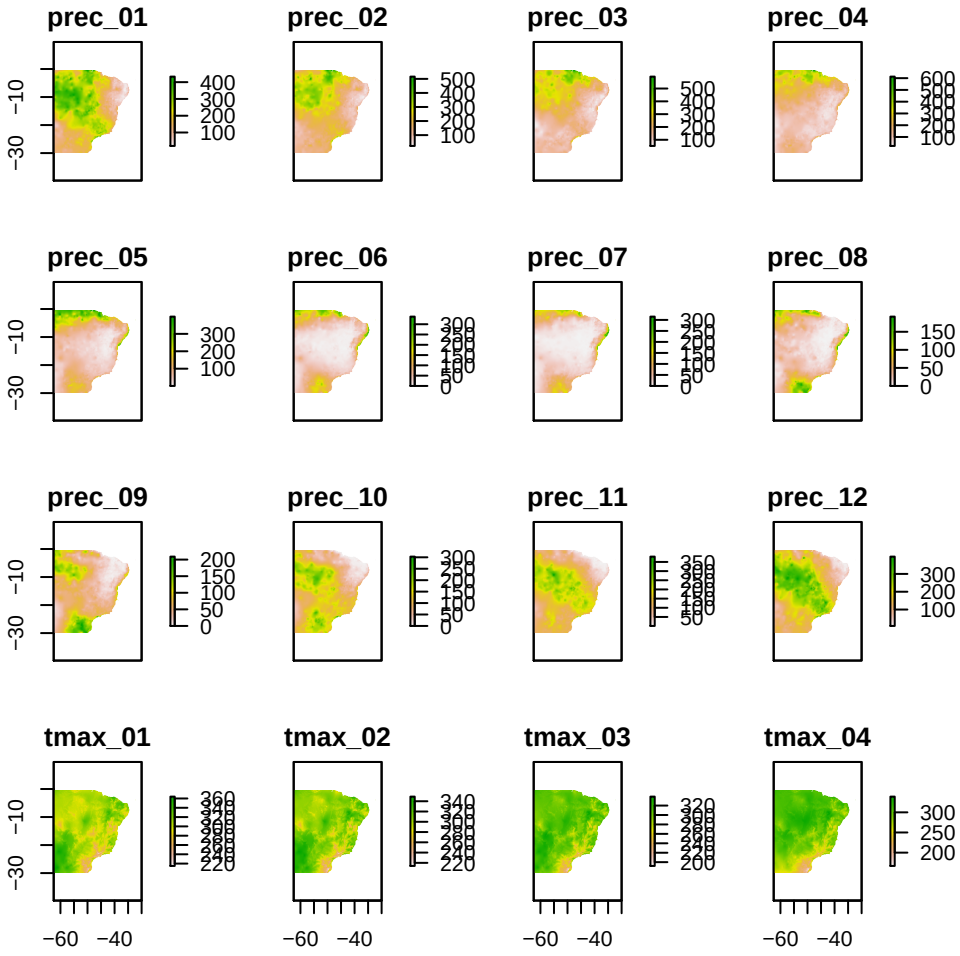
We now have an organized folder and can use function `get_rasters` to easily choose, load, organize and crop rasters in one step.

## Loading rasters

`get_rasters` will output a list of RasterStacks, with each stack representing a scenario with a layer for each variable chosen. Some of your variables will be constant accross scenarios, such as altitude. In this case you only have to write the name of the variable on your folder, with no scenario, and it will be added to every stack. The argument `raster_path` will take a path to a folder containing the rasters you want to load. You can input any raster you want, not limited to the ones you can download from WorldClim. The arguments `var` and `scenarios` allow us to choose the variables and scenarios to be loaded.

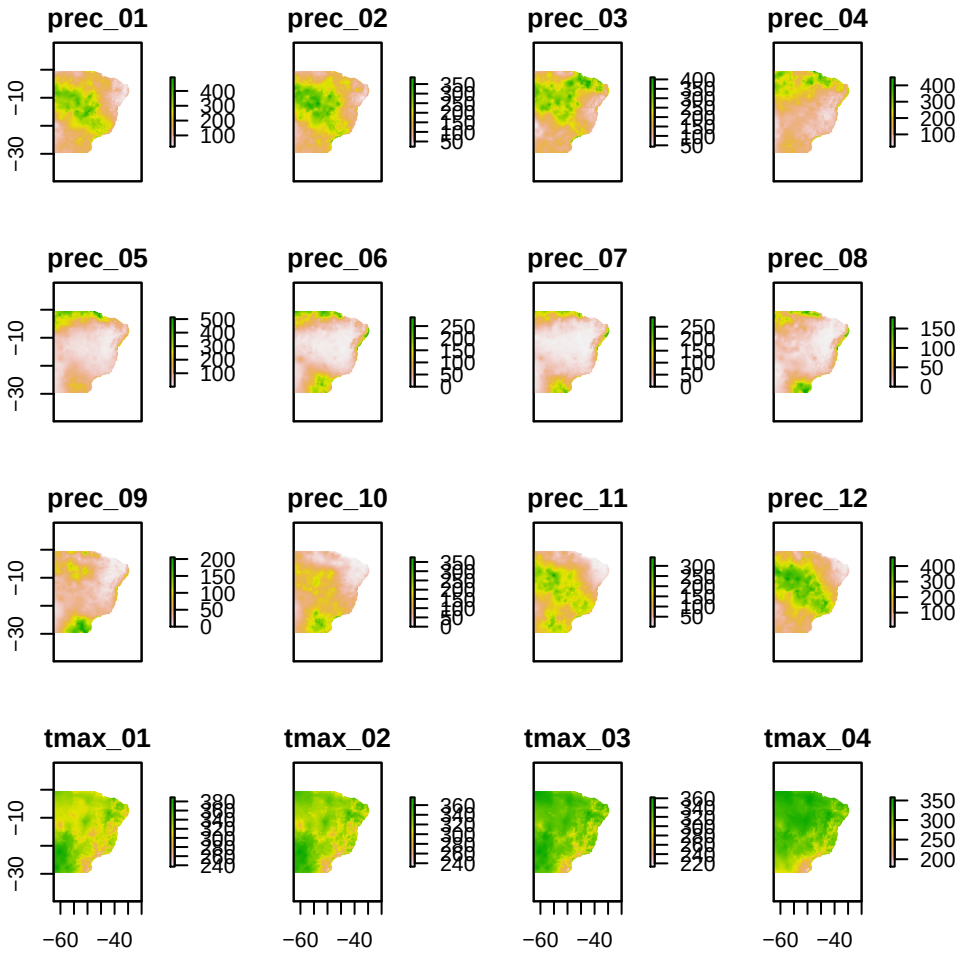
The argument `ext` tells us which is the area you desire to crop your rasters around. It accepts either a numerical vector with the coordinates of cropping limits (in order: western most longitude, easter most longitude, southern most latitude then northern most latitude), or a table of longitudes (column named `Lon`) and latitudes (column named `Lat`), such as the `TtorquatusDistribution_clean` table we created previously. In this case, the function will grab the coordinates of the most extreme points and use those as the cropping limits. The argument `margin` adds to these limits, and it is on the same unit as the coordinates you supply.

```
Ttorquatus_climate_present <-
  get_rasters(
    var = c('alt', 'prec', 'tmin', 'tmax'),
    scenario = "present",
    raster_path = paste0(mydir, "rasters/worldclim/global_rasters_10min/"),
    ext = TtorquatusDistribution_clean,
    margin = 5)

plot(Ttorquatus_climate_present$present)
```




Let's get some projections for the future years of 2050 and 2070, under different carbon emission scenarios:

```
Ttorquatus_climate_future <-
  get_rasters(
    var = c('prec', 'tmin', 'tmax'),
    scenario = c('2050rcp45', '2070rcp45', '2050rcp85', '2070rcp85'),
    raster_path = paste0(mydir, "rasters/worldclim/global_rasters_10min/"),
    ext = TtorquatusDistribution_clean,
    margin = 5)

plot(Ttorquatus_climate_future$`2050rcp45`)
```




As you can see, we have multiple rasters for each variable, one for each month. When doing biogeographical analysis, such as species distribution models, it is common to reduce those rasters to summaries for the year, such as the average or sum of each variable. Function `transform_rasters` can help us to get those summaries for the whole year or for parts of the year.

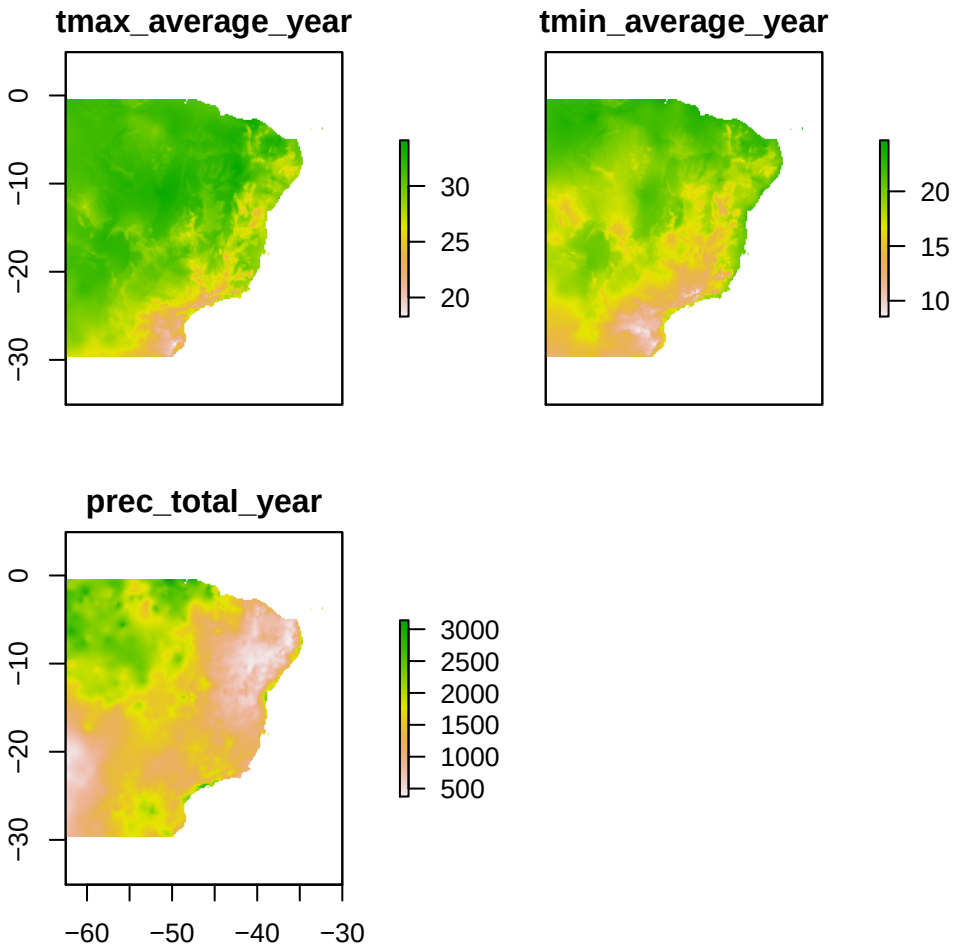
## Summarizing rasters across time

Function `transform_rasters` allows us to apply any vectorized function to any subset of rasters for each variable, so we can get measures like averages, standard deviations, totals, variance or any other summary function for the month layers. Summaries have the advantage of reducing computation time, as they allows us to work with less layers. The first argument is `raster_stack`, which is where we supply the RasterStack with the layers for the calculations. The next arguments are the new variables to be created and the functions to create them. You have to specify on those arguments the name of the variables to which the function will be applied to. There is another argument, `ncores`, which specifies the number of cores to use in parallel processing. This calculations can be pretty computationally intensive, so parallelizing can greatly reduce running time.

Let's get the average temperatures and total precipitation for the whole year, in the present:

```
Ttorquatus_present_year_summaries <-
  transform_rasters(raster_stack = Ttorquatus_climate_present$present,
                    tmax_average_year = mean(tmax/10),
                    tmin_average_year = mean(tmin/10),
```

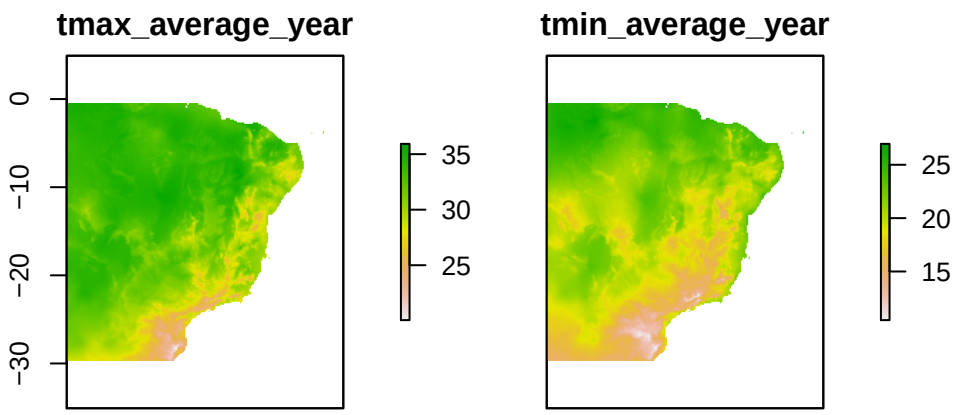
```
                    prec_total_year = sum(prec),
                    ncores = 2)

plot(Ttorquatus_present_year_summaries)
```

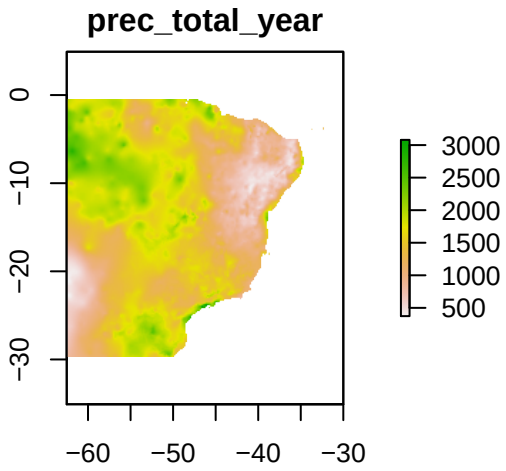



Notice we are dividing the temperatures by 10, since the original temperatures in the WorldClim files were multiplied by 10 (look at the scale in the figure above).

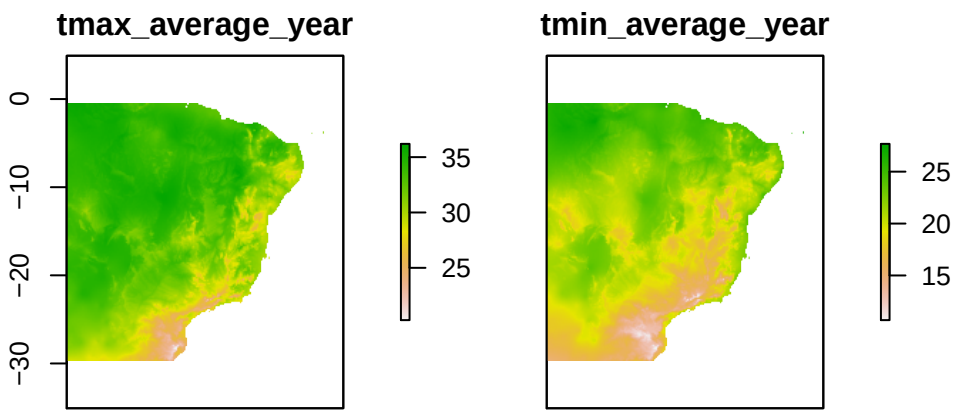
Now let's do the same summaries for the future scenarios. Since we have multiple future scenarios, stored in a list, the best way to transform them all is to use `transform_rasters` together with the function `lapply`, which applies a function to each element of a list.

```
Ttorquatus_future_year_summaries <-
  lapply(Ttorquatus_climate_future, transform_rasters,
         tmax_average_year = mean(tmax/10),
         tmin_average_year = mean(tmin/10),
         prec_total_year = sum(prec),
         ncores = 2)

plot(Ttorquatus_future_year_summaries$`2050rcp45`)
```



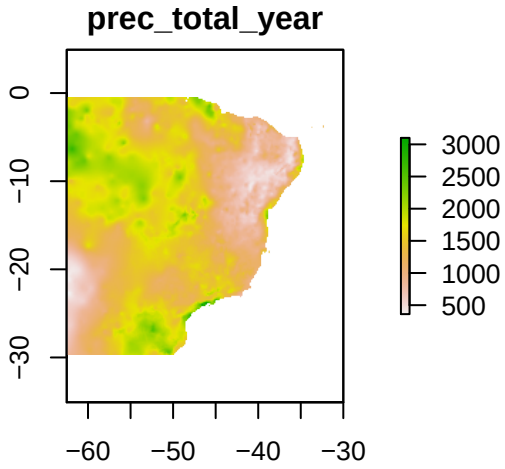




```
plot(Ttorquatus_future_year_summaries$`2070rcp45`)
```

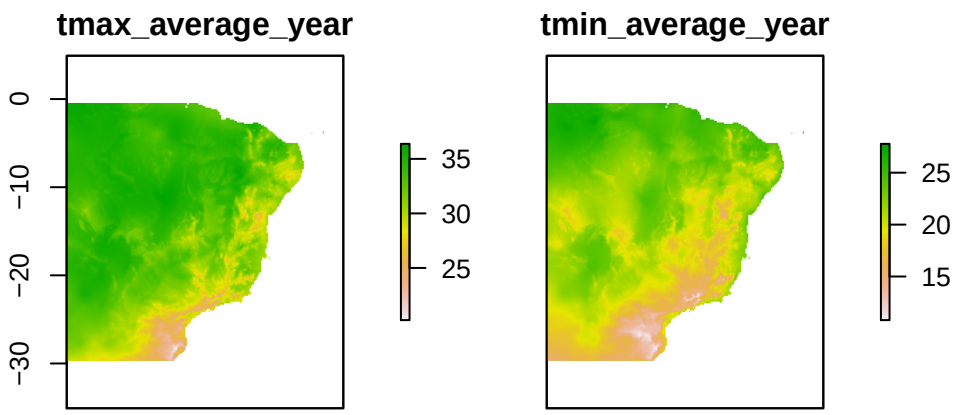






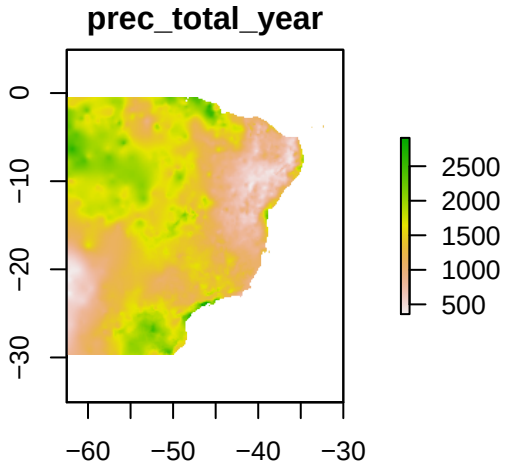



```
plot(Ttorquatus_future_year_summaries$`2050rcp85`)
```

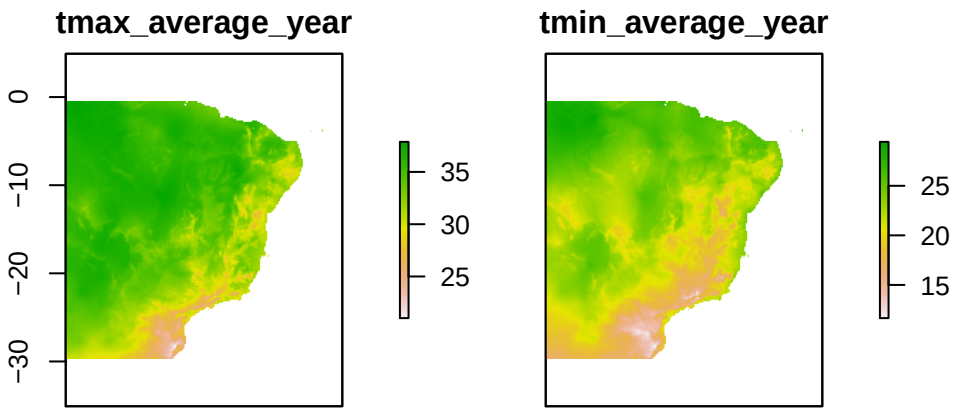


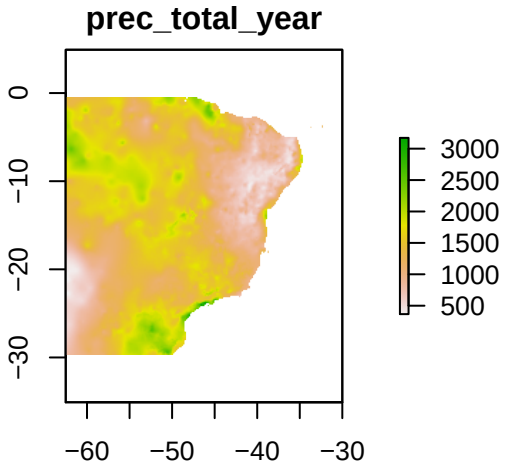







```
plot(Ttorquatus_future_year_summaries$`2070rcp85`)
```



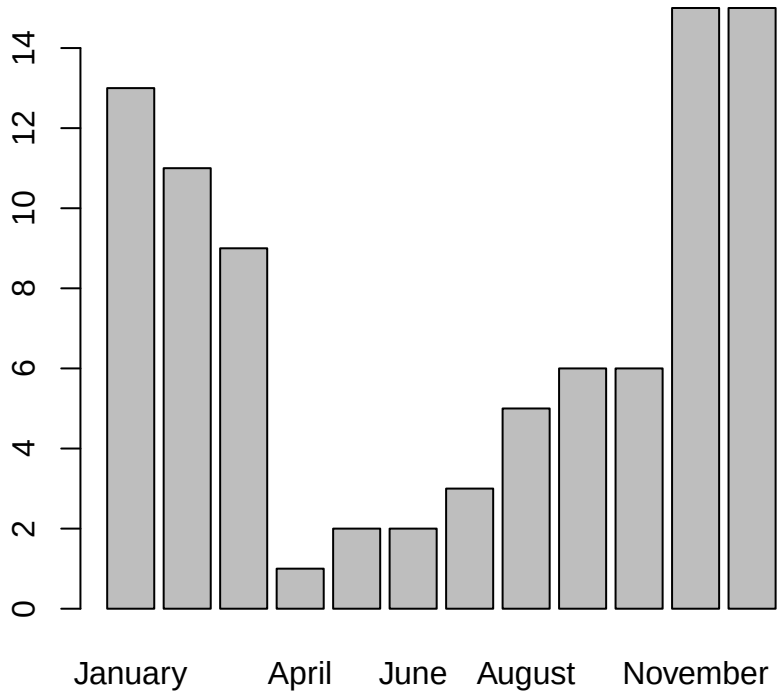

Sometimes we are interested in specific periods of the year, for example, when a phenological event such as reproduction is happening. `transform_rasters` can help us in this case as well. To illustrate this, let's separate the climatic variables according to the breeding season of *T. torquatus*. The `TtorquatusBreeding` dataset contains a binary record of the months when the lizard breeds in 16 different locations accross its distribution (Caetano et al, 2019a, in preparation). By adding up the rows of the table, we can find at which months of the year the animal is breeding in most locations.

```
TtorquatusBreeding
```

```
##          Lon        Lat January February March April May June July August
## 2  -53.87361 -29.626944       1        0     0     0   0    0    0      0
## 3  -58.74667 -27.430556       1        1     0     0   0    0    1      1
## 4  -43.52376 -23.046658       1        1     1     0   0    0    0      0
## 5  -42.84539 -22.961308       0        0     0     0   1    1    1      1
## 6  -41.53386 -22.223820       1        1     1     0   0    0    0      0
## 7  -43.59214 -21.807639       0        0     0     0   0    0    0      1
## 8  -41.02447 -21.692313       1        1     1     0   0    0    0      0
## 9  -40.96346 -21.277258       1        1     1     0   0    0    0      0
## 10 -40.44032 -20.631209       1        1     1     0   0    0    0      0
## 11 -39.85078 -18.723388       1        1     1     0   0    0    0      0
## 12 -39.22083 -17.340833       1        1     1     0   0    0    0      0
## 13 -39.07887 -16.485795       1        1     1     0   0    0    0      0
## 14 -39.09583 -16.590556       1        1     1     1   1    1    1      0
```

```
## 15 -47.91667 -15.783333       1        1     0     0   0    0    0       1
## 16 -40.01667  -7.416667       1        0     0     0   0    0    0       1
##    September October November December
## 2          1       1        1        1
## 3          1       1        1        1
## 4          0       0        1        1
## 5          1       1        1        1
## 6          0       0        1        1
## 7          1       1        1        1
## 8          0       0        1        1
## 9          0       0        1        1
## 10         0       0        1        1
## 11         0       0        1        1
## 12         0       0        1        1
## 13         0       0        1        1
## 14         0       0        1        1
## 15         1       1        1        1
## 16         1       1        1        1
```

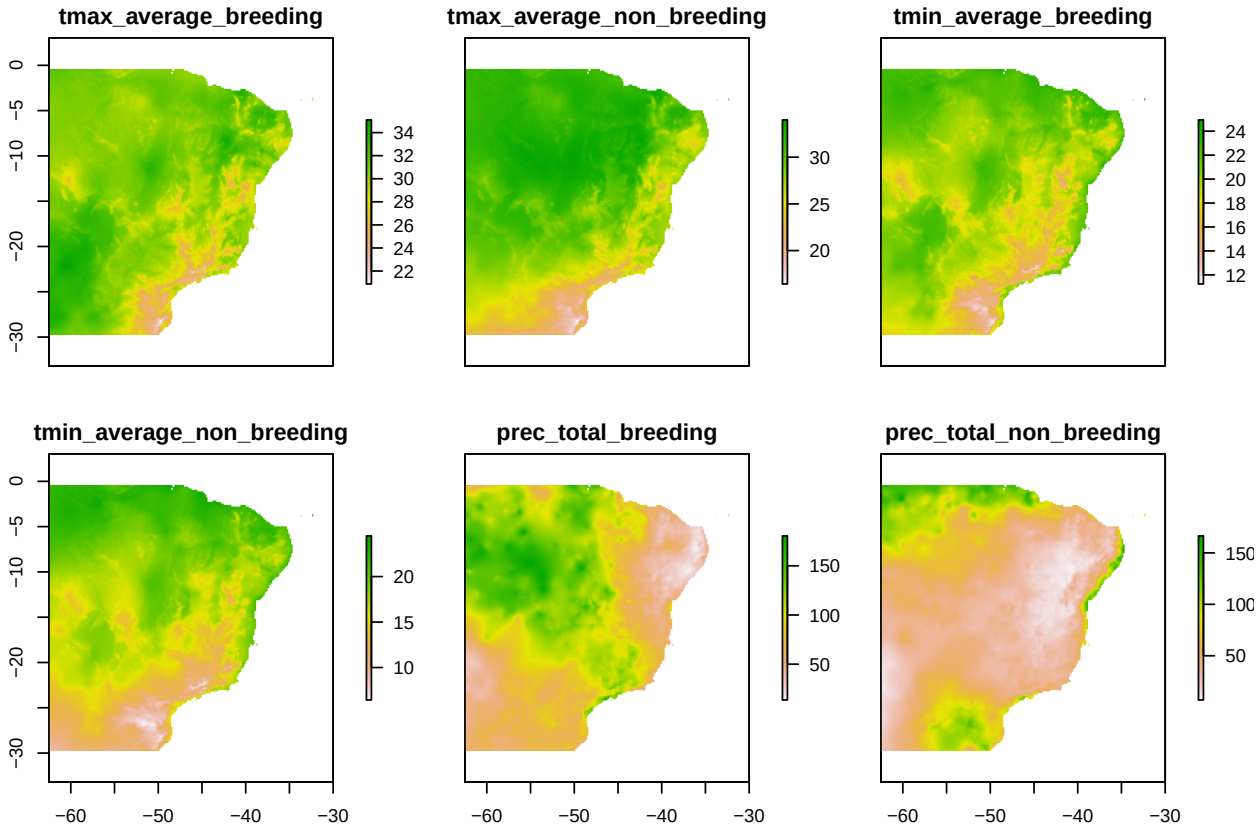
```
barplot(colSums(TtorquatusBreeding[-c(1:2)]))
```




It seems the lizard breeds from November to March in most locations. Now let's use this information and `transform_rasters` to obtain the average temperatures and total precipitation for the breeding and non-

breeding season accross the species distribution. To subset the variables we use in the calculation, we use square brackets `[`. For example, if we want to use only the layers `tmax_01`, `tmax_02` and `tmax_03` in a calculation, we should call `tmax[1:3]` on the formula of the new variable.

```
Ttorquatus_present_seasons_summaries <-
  transform_rasters(raster_stack = Ttorquatus_climate_present$present,
                    tmax_average_breeding = mean(tmax[c(1:3,11:12)]/10),
                    tmax_average_non_breeding = mean(tmax[4:10]/10),
                    tmin_average_breeding = mean(tmin[c(1:3,11:12)]/10),
                    tmin_average_non_breeding = mean(tmin[4:10]/10),
                    prec_total_breeding = sum(prec[c(1:3,11:12)]/10),
                    prec_total_non_breeding = sum(prec[4:10]/10),
                    ncores = 2)

plot(Ttorquatus_present_seasons_summaries)
```




Summary functions are an useful example, but we can apply more complex functions, such as models using organismal information, in order to create biologically relevant variables tailored for the species studied. But before we create those variables, we must fit the models in question. Function `get_predict` allows us to compare models and get a vectorized predictor function which can then easily be applied to `transform_rasters`.

## Fitting physiological models

Function `get_predict` can be used to compare models with different parameterizations or specifications for your biological processes and get you a predictor function for those models. The user needs to inform a list of models in argument `models` and the function will output a table with their AIC, BIC, log likelihood,

delta AIC, delta BIC and a rank for AIC and BIC values. The main goal of the function, however, is to create vectorized predict functions for each model, which can then be used in `transform_models` to spatially extrapolate the model. In order to make the spatial extrapolation possible, you must have spatial information for at least one of the variables on the right hand side of your model formula. The vectorized function generated can be used to get predictions of your model in contexts other than spatial, such as for time series or specific values.

We measured the maximum running speed of several individuals of *T. torquatus* under different temperatures. We also recorded their body sizes, since that variable is likely to inffluence their performance. Here is how the data look like:

```
head(TtorquatusPerformance)
```

```
##       species  id temp performance size     site
## 1 T_torquatus 209 19.9       21.87   83 Brasilia
## 2 T_torquatus 209 23.5       31.86   83 Brasilia
## 3 T_torquatus 209 28.2       24.97   83 Brasilia
## 4 T_torquatus 209 13.3        0.00   83 Brasilia
## 5 T_torquatus 209 44.4        0.00   83 Brasilia
## 6 T_torquatus 141 20.0       10.76   80 Brasilia
```

Note that we also attributed a number to each lizard to keep track of which lizard did which trial. That is important because, since the same lizard ran different trials, the data points are not independent and we have to account for that when building our model.

Argument `models` can be a single model or a list of models. In this case we are fitting GAMM models, which can account for the autocorrelation from running several tests with each individual. You can name the models by putting their names on the list. Most model algorithms that have a method for function `predict` should work. You can pass additional arguments specific for your kind of model if you wish, such as `type`, which I used below to set the scale of the prediction to the same as the response variable (see `?predict.gam`).

```
library(mgcv)

perf_no_size <-
  gamm(performance ~ s(temp, bs = 'cs'),
       random = list(id = ~ 1),
       data = TtorquatusPerformance)

perf_size <-
  gamm(performance ~ s(temp, bs = 'cs') + size,
       random = list(id = ~ 1),
       data = TtorquatusPerformance)

perf_functions <-
  get_predict(models = list(perf_s = perf_size$gam,
                            perf_ns = perf_no_size$gam),
              type = "response")
```

```
## Warning in min(AIC): no non-missing arguments to min; returning Inf
```

```
## Warning in min(BIC): no non-missing arguments to min; returning Inf
```

```
## [1] logLik  AIC     BIC     dAIC    dBIC    rankAIC rankBIC
## <0 rows> (or 0-length row.names)
```

The function prints a table containing statistics for comparing the models, the log likelihood, AIC, BIC, delta AIC, delta BIC, (which are the all the AICs and BICs subtracted by the smallest one) and ranks for AIC and BIC, from smallest to largest. As we can see, the model not considering size was the best one, by a small margin. We can now take the predictor functions from the `perf_functions` list and use them to predict the running speed of the lizard in different temperatures, and in case of the model considering size, of different sized lizards.

```
perf_nsFUN <- perf_functions$perf_ns
perf_sFUN <- perf_functions$perf_s

perf_nsFUN(temp = 20)
```

```
## [1] 10.68421
```

```
perf_nsFUN(temp = 30)
```

```
## [1] 17.52461
```

```
perf_nsFUN(temp = 40)
```

```
## [1] 1.347293
```

```
perf_sFUN(temp = 30, size = 30)
```

```
## [1] 16.45186
```

```
perf_sFUN(temp = 30, size = 70)
```

```
## [1] 17.36206
```

```
perf_sFUN(temp = 30, size = 100)
```

```
## [1] 18.04472
```

Those functions are vectorized, which means that you can apply them to vectors of values, and they will return a vector of results of the same length.

```
perf_nsFUN(temp = 30:35)
```

```
## [1] 17.52461 17.62255 18.10986 18.62222 18.62435 17.58064
```

```
perf_sFUN(temp = 30, size = 70:75)
```

```
## [1] 17.36206 17.38482 17.40757 17.43033 17.45308 17.47584
```

```
perf_sFUN(temp = 30:35, size = 70:75)
```

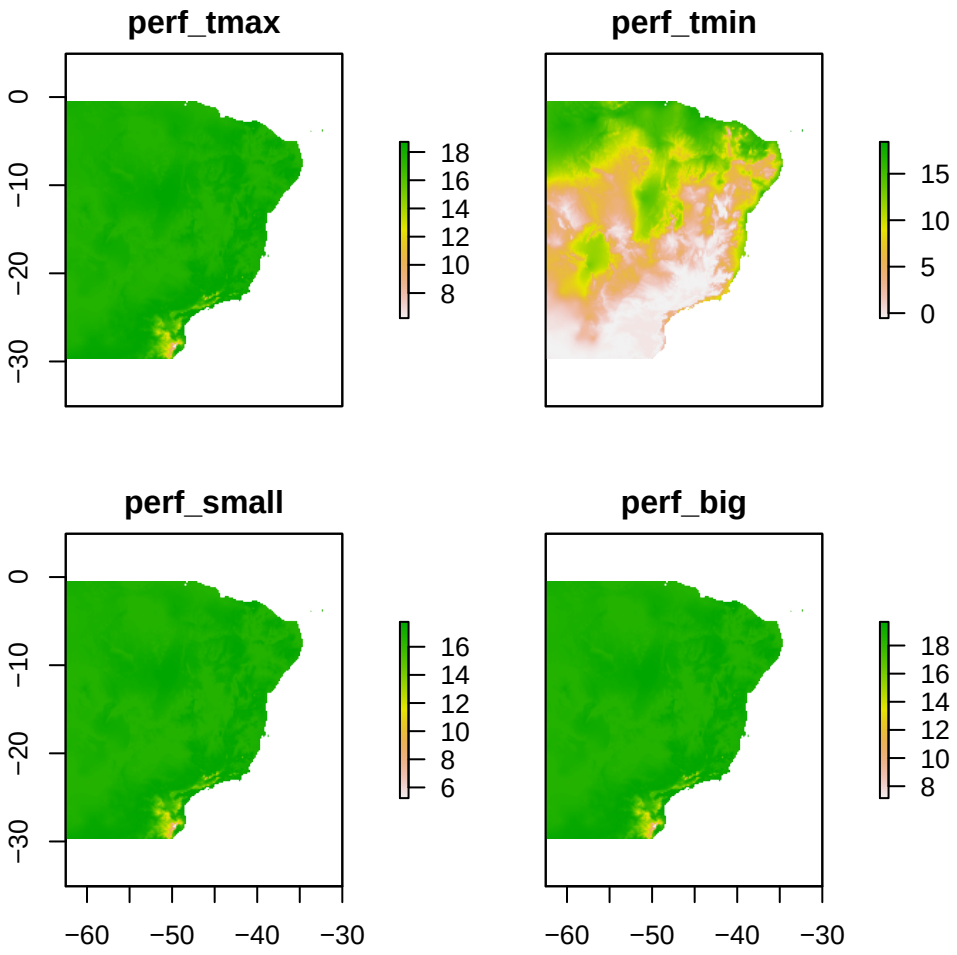
```
## [1] 17.36206 17.48891 18.01427 18.56994 18.61472 17.60705
```

And because they are vectorized they can be applied to rasters of the predictor variables using function `transform_rasters`, allowing us to spatialize the model and get predictions of the response variable for our study area.

## Spatializing a physiological model

We can use our new functions just like we used the summary functions in `transform_rasters`. We can apply them to different measures of temperatures, or in the case of the function considering size, assign different size values. This allows us to generate many different kinds of spatial information for the species.

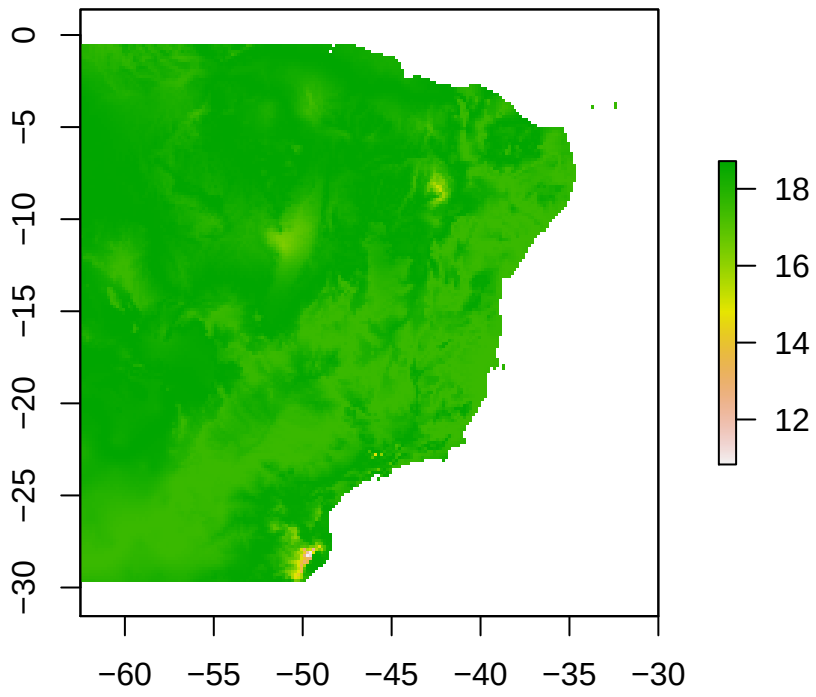
```
Ttorquatus_present_perf <-
  transform_rasters(raster_stack = Ttorquatus_present_year_summaries,
                    perf_tmax = perf_nsFUN(temp = tmax),
                    perf_tmin = perf_nsFUN(temp = tmin),
                    perf_small = perf_sFUN(temp = tmax, size = min(TtorquatusPerformance$size)),
                    perf_big = perf_sFUN(temp = tmax, size = max(TtorquatusPerformance$size)),
                    ncores = 2)

plot(Ttorquatus_present_perf)
```



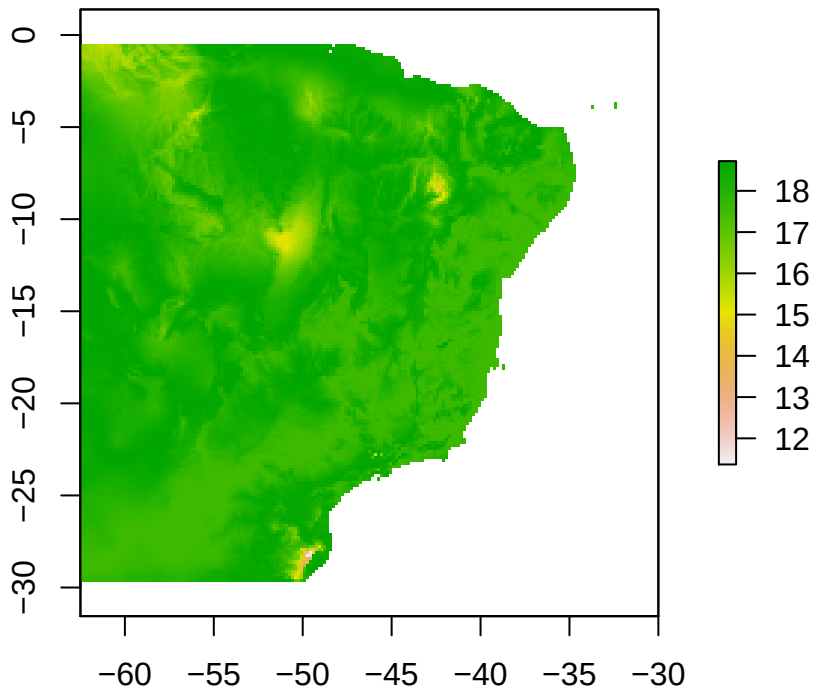

As we can see, including size differences does not generate very different predictions. That is to be expected, since the model including size did not perform very different from the model not including it.

We can apply our models to many different scenarios using `lapply`, in the same way we did before with the summary functions:

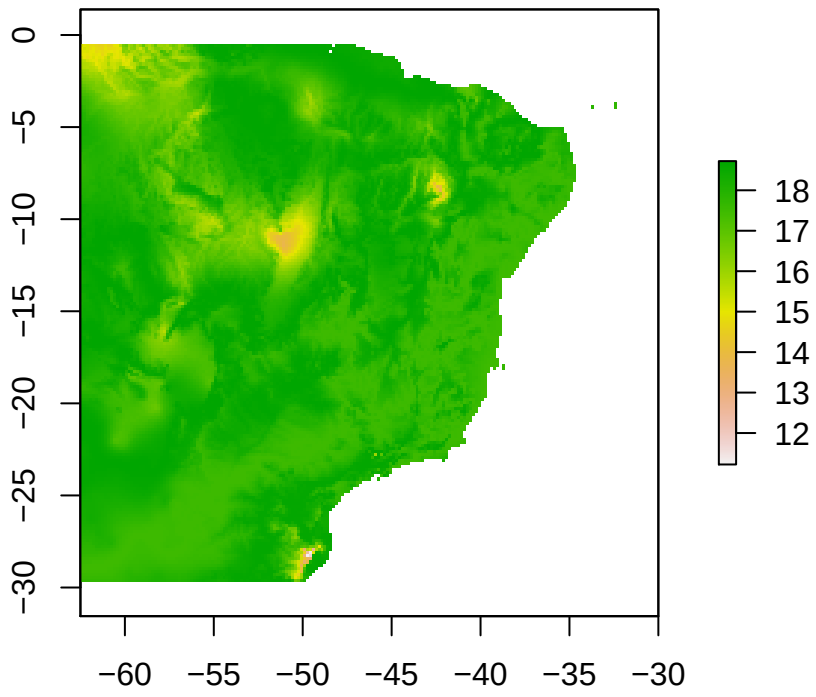
```
Ttorquatus_future_perf <-
  lapply(Ttorquatus_future_year_summaries,
         transform_rasters,
         perf_tmax = perf_nsFUN(temp = tmax),
         ncores = 2)

plot(Ttorquatus_future_perf$`2050rcp45`)
```



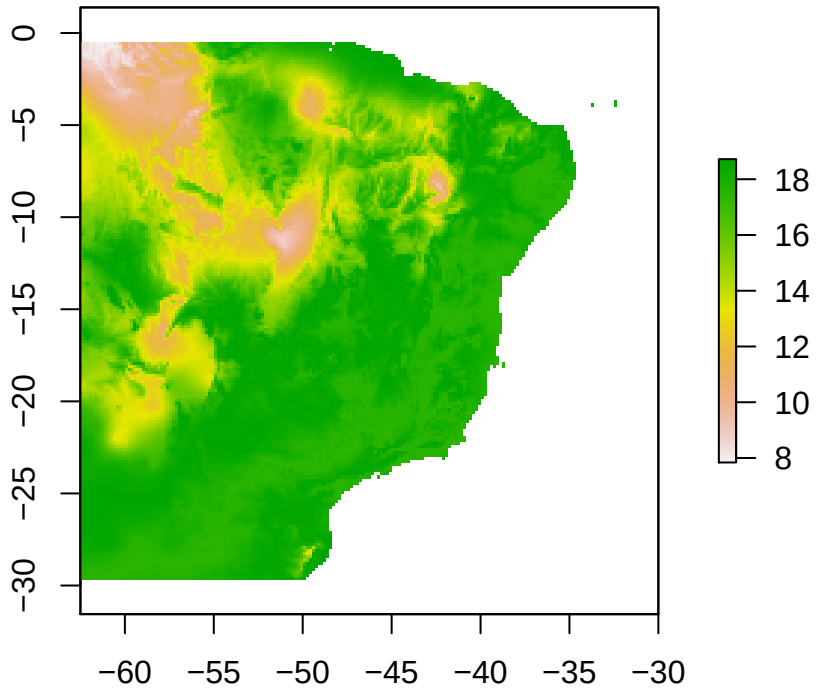

```
plot(Ttorquatus_future_perf$`2070rcp45`)
```




```
plot(Ttorquatus_future_perf$`2050rcp85`)
```




```
plot(Ttorquatus_future_perf$`2070rcp85`)
```




Those are the basic functions provided by Mapinguari, which can be used creatively to generate many kinds of spatial information that can be useful in your biological analysis. In the next chapters, I will show examples of other variables that can be created with those functions, as well as more specific functions provided by the package.

## Time of activity

Sinervo et al, 2010, implemented a simple model to estimate daily activity duration for many species of lizard worldwide and used these data in a extinction prediction model. In this chapter, we are going to use Mapinguari to do the similar estimates using three different methods.

Estimating temperature dependent time of activity requires two other estimates: the daily temperature variation to which animals experience and the range of temperatures in which they are active. The methods presented here differ in the estimation of daily temperature variation. For the estimates of temperature range of activity we are going to use the methods tested by Caetano et al 2019b.

We have measure the range of temperatures lizards prefer to be by placing them on temperature gradients, which are lanes where the lizard is contained and which has a hot side and a cold side, and the lizard is free to choose which temperature in between those extremes it prefers to be. Its body temperature is collected by a datalogger every minute for one hour (see details in Caetano et al 2019b). Here is the data frame containing this data:

```
head(TtorquatusGradient)
```

```
##   temp id           site
## 1 29.7  0 Nova_Xavantina
## 2 29.9  0 Nova_Xavantina
## 3 29.7  0 Nova_Xavantina
## 4 29.9  0 Nova_Xavantina
## 5 30.0  0 Nova_Xavantina
## 6 30.2  0 Nova_Xavantina
```

We are going to remove temperature outliers and calculate two metrics: `vtmin`, which is the 5% percentile temperature lizards experienced in the gradient and `vtmax`, the 95% percentile temperature. This interval generates the best predictions for this species (Caetano et al, 2019b). If you are doing the same estimates for a different species, you should carefully consider which interval to use.

```
outliers_gr <- boxplot(TtorquatusGradient$temp, plot=FALSE)$out
TtorquatusGradient_no <- TtorquatusGradient[-which(TtorquatusGradient$temp %in% outliers_gr),]

vtmin <- quantile(TtorquatusGradient_no$temp, 0.05)
vtmax <- quantile(TtorquatusGradient_no$temp, 0.95)

data.frame(vtmin, vtmax)
```

```
##    vtmin   vtmax
## 5%  20.8 38.6705
```

## Sinusoid method

Now that we have estimated our temperature range, we are going to apply it to estimates of daily temperature variation. First, let's start with the model used by Sinervo et al, 2010. It consists in simulating the daily temperature variation as a sine wave ranging from the minimum to the maximum daily temperatures registered for the day. This method has the advantage of being simpler and thus requiring less computation, and its results can be comparable to more complex methods (Caetano et al, 2019b). Mapinguari has a built-in function, `sin_h` to perform those calculations. Argument `tmax` indicates the maximum air temperature of the day, `tmin`, the minimum, `thrs` is the temperature threshold above which time is accounted, and `res` is the number of hour fractions to be included on the time estimates.

So let's see for how many hours daily temperatures are above `vtmin` in a day when temperatures ranged from 20 to 40 degrees celsius:

```
hours_above_vtmin <-
sin_h(tmax = 40,
      tmin = 20,
      thrs = vtmin,
      res = 3)

hours_above_vtmin
```

```
## [1] 21
```

Above `vtmax`:

```
hours_above_vtmax <-
sin_h(tmax = 40,
      tmin = 20,
      thrs = vtmax,
      res = 3)

hours_above_vtmax
```

```
## [1] 3.666667
```

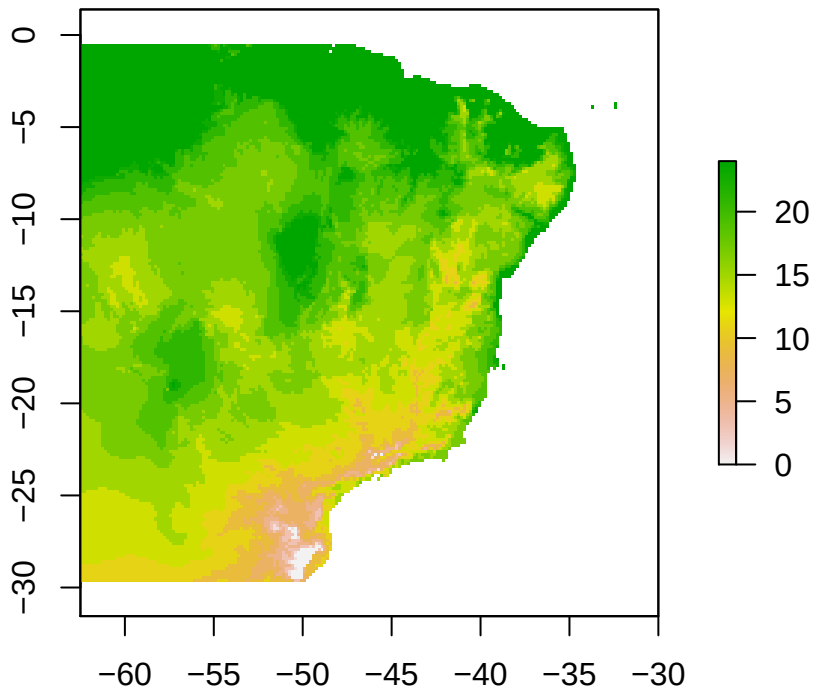
And to see the amount of time between them, just subtract the estimates for the higher temperature from the estimates for the lower temperature:

```
hours_above_vtmin - hours_above_vtmax
```

```
## [1] 17.33333
```

To estimate time of activity for *T. torquatus* distribution, simply use `transform_rasters` to apply `sin_h` over temperature rasters:

```
# fix this so you can subtract ftmin - ftmax, vtmin - vtmax on the argument
Ttorquatus_ha_sin <-
  transform_rasters(Ttorquatus_present_year_summaries,
                    ha_vtmin = sin_h(tmax = tmax, tmin = tmin, thrs = vtmin, res = 1),
                    ha_vtmax = sin_h(tmax = tmax, tmin = tmin, thrs = vtmax, res = 1),
                    ncores = 2)

ha_vt <- Ttorquatus_ha_sin$ha_vtmin - Ttorquatus_ha_sin$ha_vtmax

plot(ha_vt)
```




## Operative Method

Alternatively, we can use actual measures of daily microclimatic variation performed at microhabitats used by *T. torquatus*. This method might add realism, as it considers microhabitats used by the species, but it requires extrapolation from a few locations where measures were taken to a very large area, which might be affected by unmeasured sources of variation. We have measured those temperatures using operative temperature models at 6 sites across the species range, and those records are on column `temp`. Also included are maximum daily air temperaturess recorded at nearby weather stations for every day sampled, at column `t_air_max`. Let's take a look at the structure of this data set:

```
head(TtorquatusOperative)
```

```
##       site description       Lon       Lat   temp microhabitat year month
## 1 Brasilia      forest -47.88278 -15.79389 22.633         tree 2014     5
## 2 Brasilia      forest -47.88278 -15.79389 22.609         tree 2014     5
## 3 Brasilia      forest -47.88278 -15.79389 22.633         tree 2014     5
## 4 Brasilia      forest -47.88278 -15.79389 22.609         tree 2014     5
## 5 Brasilia      forest -47.88278 -15.79389 22.609         tree 2014     5
## 6 Brasilia      forest -47.88278 -15.79389 22.609         tree 2014     5
##   day hour minute t_air_max
## 1   8   12      0    28.772
```

```
## 2   8   12      5    28.772
## 3   8   12     10    28.772
## 4   8   12     15    28.772
## 5   8   12     20    28.772
## 6   8   12     25    28.772
```

There is some inconsistence in the time resolution of the records, some recorded temperatures every minute, others every five minutes, other every ten minutes. In other to standardize measures, we can convert all to one hour resolution using package `dplyr`, by averaging columns `temp` and `t_air_max` for every hour.

```
op_hour <-
  TtorquatusOperative %>%
  dplyr::group_by(site, description, microhabitat, year, month, day, hour) %>%
  dplyr::summarise(temp = mean(temp),
                   t_air_max_h = mean(t_air_max))
```

We can use a custom function to calculate in which hours temperatures were between `vtmin` and `vtmax`:

```
hvtFUN <- function(x) ifelse(x > vtmin & x < vtmax, 1, 0)

op_ha <-
  op_hour %>%
  dplyr::mutate(hvt_h = hvtFUN(temp))
```

Now, we can adress the microclimatic variation in the data, by using different functions to summarise time of activity estimates between habitats. For example, we can assume animals will use microhabitats randomly, regardless of their thermal quality, and create variable `hvt_mh`, which is the average time of activity between microhabitats at that hour. alternatively, we can assume animals will always seek thermally adequate environments and create variable `hvt_tr_mh`, which takes the maximum value between available microhabitats at any given time. We include the average `t_air_max` to preserve its values in the table:

```
op_mh <-
  op_ha %>%
  dplyr::group_by(site, description, year, month, day, hour) %>%
  dplyr::summarise(hvt_mh = mean(hvt_h),
                   hvt_tr_mh = max(hvt_h),
                   t_air_max_mh = mean(t_air_max_h))
```

Now, we can sum hours of activity for each day generated under those two assumptions:

```
op_day <-
  op_mh %>%
  dplyr::group_by(site, description, year, month, day) %>%
  dplyr::summarise(hvt = sum(hvt_mh),
                   hvt_tr = sum(hvt_tr_mh),
                   t_air_tp = mean(t_air_max_mh))
```

Now we can model the relationship between time of activity estimated under different assumptions and air temperatures measured at weather stations, using logistic regressions:

```
# Fit simple logistic models (Richard's wouldn't fit all)
hvt_logistic <- nls(hvt + 0.00001 ~ SSlogis(t_air_tp, Asym, xmid, scal), data = op_day)

hvt_tr_logistic <- nls(hvt_tr + 0.00001 ~ SSlogis(t_air_tp, Asym, xmid, scal), data = op_day)
```

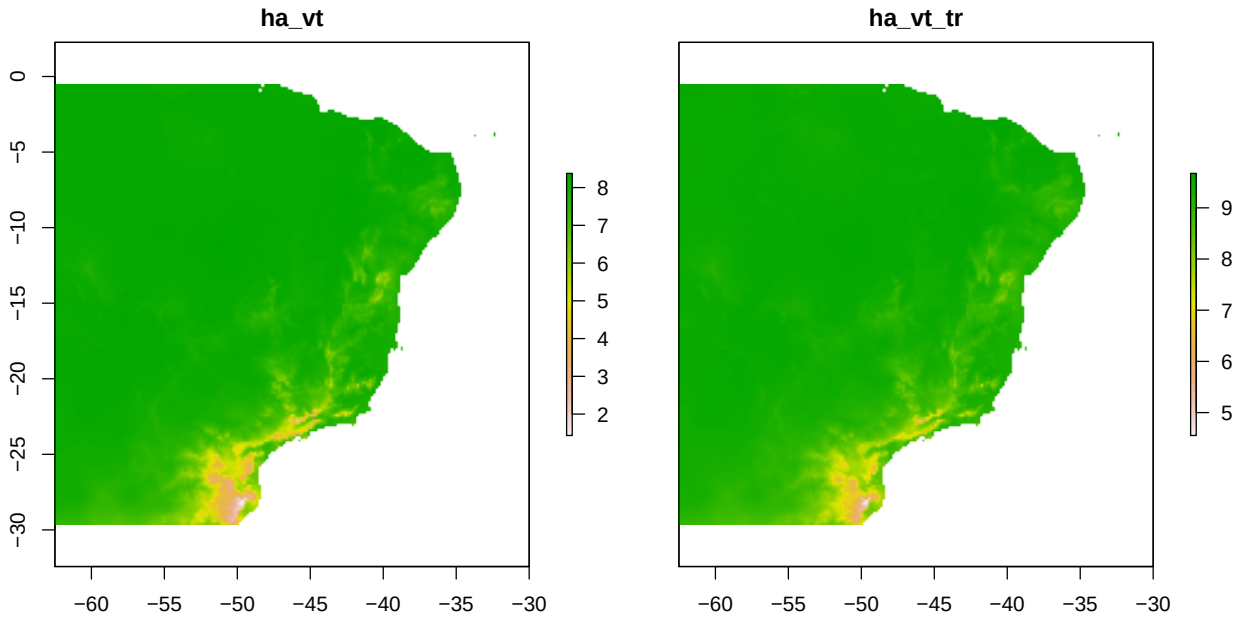
And them use `get_predict` to generate predictor functions:

```
pred_ha <-
  get_predict(list(hvtFUN = hvt_logistic,
                   hvt_trFUN = hvt_tr_logistic))
```

```
##               logLik      AIC      BIC     dAIC     dBIC rankAIC rankBIC
## hvtFUN    -278.7375 565.4750 577.4368  0.00000  0.00000       1       1
## hvt_trFUN -284.6771 577.3543 589.3160 11.87923 11.87923       2       2
```

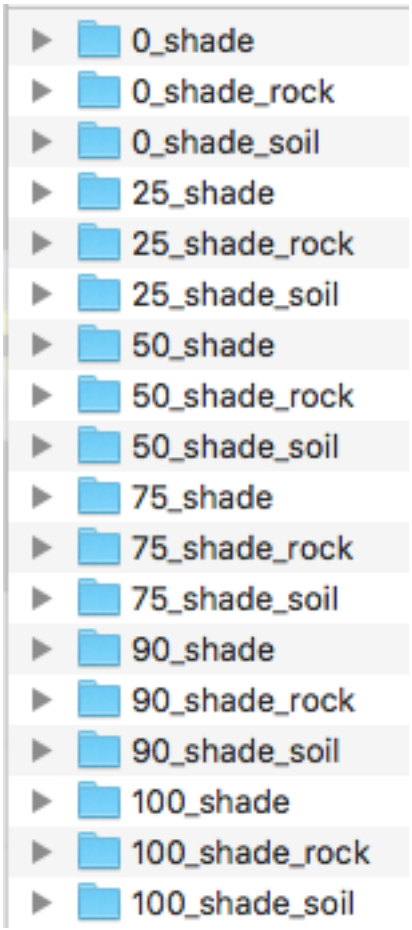
While we lose some information with this simplification, that allows us to easily spatially extrapolate time of activity estimates over air temperature rasters using function `transform_rasters`, similar to what we have done previously:

```
Ttorquatus_ha_operative <-
  transform_rasters(Ttorquatus_present_year_summaries,
                    ha_vt = pred_ha$hvtFUN(t_air_tp = tmax),
                    ha_vt_tr = pred_ha$hvt_trFUN(t_air_tp = tmax),
                    ncores = 2)

plot(Ttorquatus_ha_operative)
```

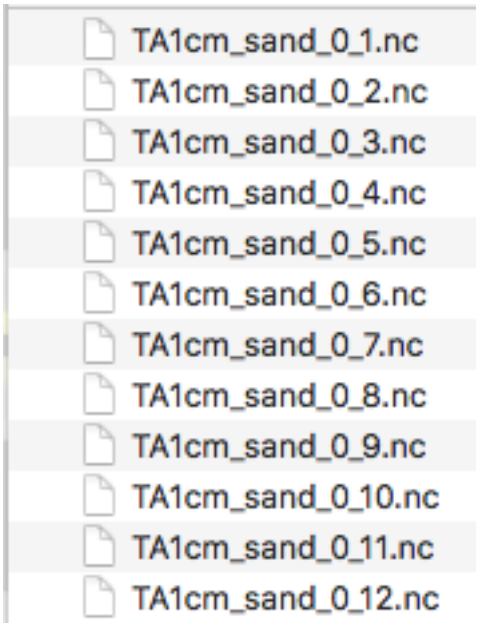



## Microclim method

A third method is to obtain microclimatic temperatures from biophysical models. We use the microclim data set created by Kearney et al 2014. Here is how I have organized this data in my folder, with the rasters for each microhabitat in separate folders:




Each folder contains 12 raster files, one for each month of the year, and each file contains 24 layers, one for each hour of the day:




We can then use another Mapinguari function, `multi_extract`, to easily extract the temperature values from multiple rasters in those folders and generate a microclimate table similar to the one used before on the

operative method. You can use argument `layers` to select which layers (hours) you want to include. Here we will select only the hours in which there is daylight at the region, from 6 am to 6 pm. You can also use arguments `folders` and `files` to select which folders (microhabitats) and which files (months) to include. If those arguments are not included, all folders, files and layers will be selected:

```
Ttorquatus_microclim <-
  multi_extract(raster_path = paste0(mydir, "rasters/microclim"),
                coord = TtorquatusDistribution_clean[-1],
                layers = 6:18)
```

```
##
##
## folder: 0_shade
##
## folder: 0_shade_rock
##
## folder: 0_shade_soil
##
## folder: 100_shade
##
## folder: 100_shade_rock
##
## folder: 100_shade_soil
##
## folder: 25_shade
##
## folder: 25_shade_rock
##
## folder: 25_shade_soil
##
## folder: 50_shade
##
## folder: 50_shade_rock
##
## folder: 50_shade_soil
##
## folder: 75_shade
##
## folder: 75_shade_rock
##
## folder: 75_shade_soil
##
## folder: 90_shade
##
## folder: 90_shade_rock
##
## folder: 90_shade_soil
```

We then use dplyr to estimate time of activity as we have done before for the operative data:

```
mc_hour  <-
  Ttorquatus_microclim %>%
  dplyr::mutate(hvt_h = hvtFUN(value))
```

```
mc_mh <-
mc_hour %>%
  dplyr::group_by(Lon, Lat, file_ind, layer) %>%
  dplyr::summarise(hvt_mh = mean(hvt_h),
                   hvt_tr_mh = max(hvt_h))

mc_day <-
mc_mh %>%
  dplyr::group_by(Lon, Lat, file_ind) %>%
  dplyr::summarise(hvt = sum(hvt_mh),
                   hvt_tr = sum(hvt_tr_mh))
```

These estimates can now be spatially extrapolated as done before for the operative data or in any other statistical analysis you might be interested in.

# References


Caetano, Gabriel H. O., Santos, J. C., Miles, D. B., Colli, G. R., & Sinervo, B. (2019). "Drivers of reproductive seasonality in two tropical lizards" in preparation.

Caetano, Gabriel H. O., Santos, J. C., Godinho, L. B., Cavalcante, V. H. G. L., Diele-Viegas, L. M., Campelo, P. H., Martins, L. F., Oliveira, A. F. S., Alvarenga, J. M., Wiederhecker, H. C., Novaes e Silva, V., Werneck, F. P., Miles, D. B., Colli, G. R., Sinervo, B. (2019). "Is time of activity a good predictor for ectotherms distributions?". in preparation.

Hijmans, Robert J., Cameron, S. E., Parra, J. L., Jones, P. G., & Jarvis, A. "Very high resolution interpolated climate surfaces for global land areas." International Journal of Climatology: A Journal of the Royal Meteorological Society 25.15 (2005): 1965-1978.

Kearney, Michael R., Andrew P. Isaac, and Warren P. Porter. "microclim: Global estimates of hourly microclimate based on long-term monthly climate averages." Scientific data 1 (2014): 140006.

Sinervo, Barry, Mendez-De-La-Cruz, F., Miles, D. B., Heulin, B., Bastiaans, E., Villagrán-Santa Cruz, M., ... & Gadsden, H. (2010). "Erosion of lizard diversity by climate change and altered thermal niches." Science 328.5980 (2010): 894-899.