INTERNSHIP REPORT

Biologging database on seabirds: formatting and development of a Shiny platform for the exploration

May 2022 - August 2022

Internship supervisor : Dr. Sophie LANCO BERTRAND & Dr. Amedee ROY

IRD, Sète

Author : Marie JOIGNEAU, student at Institut Agro Rennes Angers, gap year in agronomy

Referent teacher : Dr. Marie-Pierre Etienne

# **Introduction**

**(Tropical sea bird conservation)** Brazil has 8500km of coastline and 4.5 million of km² of Exclusive Economic Zone *(Carlos and al., 2009)*. A EEZ is an area of the sea in which a sovereign state has special rights regarding the exploration and use of marine ressources *(United Nations, 2011)*. The same can be said for Perou with a EEZ of 906,000 km². Because of that, Brazil and Perou have several types of sea bird species and has responsibility in their conservation *(Carlos and al., 2009)*. Indeed, in term of marine islands providing breeding and nesting, Brazil has, among other, the Abrolhos, Fernando de Noronha, São Pedro and São Paulo archipelago, and Perou the Pescadores and Guanape archipelagos.

**(Understanding of ecosystems)** Also, beyond conservation, the behavior of these birds, as marine predators, is influenced by the evolution of marine ecosystems and is so a way to watch and evaluate them. Studying seabird permits to estimate the variability of oceanic ecosystems due to climate change *(Gremillet and Boulinier, 2009)*.And so, all the information about birds is really important for defining conversation strategies, protecting species, and guarantying ecosystems’ stability *(Croxall and al., 2012)*.

**(The TABASCO JEAI project)** It’s in this context that the TABASCO JEAI project takes place. This project is focused on the reinforcement of the research on the ecology of the tropical seabirds in Brazil. There are three main thematics: (1) Biogeography and populations connectivity, (2) Behavioral and trophic ecological niches and (3) tropical seabird conservation and Marine Spatial Planning. The JEAI support fieldwork activities in five tropical insular ecosystems targeting six study species. The study sites provide an ideal setting of gradients along latitude, distance to the coast and level of anthropic pressures to develop comparative approaches for identifying the drivers of the health of the seabird population and identify the variety of conservation challenges that should be considered into the governance policy of those areas *(Tavares Nunes G. & Bertrand S., 2019).*

**(TABASCO JEAI goal)** As in partnership with IRD, the French National Research Institute for Sustainable Development, this project target 3 Sustainable Development Goals: fight against climate change, quality education and aquatic life.

**(My project)** Thanks to previous fieldworks, we have gathered a lot of sea bird data divided in time depth recorder data and gps data. GPS data is divided in different models - GIPSY, i-gotU, MiniGPSlog and Axy-trek – presented on different format – excel, text – and in different ways – different columns, event lines … -. The first part of my work is to format all that data to be able to read and compare them. I focus 4 variables that are in common in all files: date, time, latitude and longitude of the bird. I will then obtain news variables – step speed, distance to the colony …- used for later studies.

Then I will work on a Shiny App, used to better visualize dataset and to facilitate the work of biologists. I will add improvements to this app to make it adapted to the adding of new datasets by the new fieldworks coming.

# **Formatting dataset : program explanation**

## **1. Main program**

The main program contains all the necessary steps only done by the computer to format the dataset. All the necessary steps, including the manual ones, are fully explained in another part of the report.

### **a) Find sub folders**

We set the folder “folders to format” as the mother folder. The idea is that it’s possible to put several folders in this mother folder to be formatted, like the one already put named “files to format”.

Usually, I just format the files group by group only in the only and first sub folder “files to format”. It’s easier to control if there is a problem.

### **b) Separate ID (if needed)**

Some files contain several trajectories coming from different birds, and so different ID. This part uses the main function “format\_data\_in\_a\_folder” and permit to obtain sub files, separated by the ID.

### **c) Rename files**

This part permits to rename trajectory files.

First, we fill the metadata file with the names of the files. For that we need to open the metadata file. Then to put all the trajectory files in “files to format”. All the names of the files will go to the metadata column “file\_original”. Metadata file need to not be in the same folder as trajectory files, or else it will be considered as a file and noted in the column. That’s why it is saved in “metadata to format” folder.

Then we save the metadata and complete the other columns except the last 2.

Then we fill the "file\_renamed\_by\_Marie" and "file\_formatted\_by\_Marie" columns and save a last time the metadata complete.

Finally, we rename the raw files using the "file\_renamed\_by\_Marie" column.

### **d) Problem Axy-trek**

Axy-trek files contains 3 devices giving data at the same time: the accelero with x,y,z position every 50Hz (50 lines by second), the TDR (Time Depth Recorder) every 1 Hz (1 line by second), and the GPS with latitude and longitude positions every 10 sec. This part permits to take care of Axy-trek files, taking just the lines with GPS data that is interesting for this study.

Due to the huge number of lines in this type of files, I cut them in several small files, using a external device.

To filter it, there is 3 cases depending of the Axy-trek format. The first case concerns Axy-trek files with space between columns and not always value for some columns, with also not GPS values in each line, resulting on a difficult opening in R. In the second case, we have gps data in all lines of the Axy-trek and so too many lines with 50 positions each second. And the last case is the ideal Axy-trek file with sometime several lines in 1 second.

Finally, I merge the small files into a full trajectory file of the bird.

### **e) Format**

This is the last part where we format the file. The following functions explained are used here.

## **2. Functions**

### **a) set\_parameter**

**Title:** set\_parameter

**Description:** permit to set parameters for the following code

**Usage:** set\_parameter()

**Argument:** /

**Details:**

The goal is to obtain a vector with all the parameters which will be used for other functions of the program.

There are 2 choices : either the user has already entered them in the csv “parameters - already\_chosen” and he calls them, or he enters them one by one, following the indications.

**Values:** param\_vec (a vector containing all the parameters necessary for the entire processus)

### **b) open\_file**

**Title:** open\_file

**Description:** permit to open the file even if exception

**Usage:** open\_file(full\_path,param).

**Argument:**

* full\_path (path of the file)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

Permit to open the file, depending of its category (csv, txt) and if there are exceptions (extra rows, lines with for example only particular words like EVENT …).

**Values:** df (dataframe)

### **c) rename\_and\_order\_file**

**Title:** rename\_and\_order\_file

**Description:** keep only date, time, longitude, and latitude columns from the original dataframe, or rename id column

**Usage:** rename\_and\_order\_file(df,param,separ).

**Argument:**

* df (dataframe)
* param (vector of parameters chosen by user in set\_parameter function)
* separ (if data needs to be separated) = “separate” ,"no\_separate"

**Details:**

If the dataset doesn’t need to be separated: the new dataframe takes only the date, time, longitude, and latitude from the original dataframe by the column number informed by user in the vector param. Also, if the latitude and longitude are written with N/S/E/W and not with +/-, it needs to be corrected. It’s informed by the user in the vector param.

If the dataset needs to be separated : the ID column is renamed. It permits later to find it for splitting the dataset by the ID in split\_by\_ID\_and\_save function.

**Values:** df (dataframe)

### **d) split\_by\_ID\_and\_save**

**Title:** split\_by\_ID\_and\_save

**Description:** split the dataframe by ID and save each one of them

**Usage:** split\_by\_ID\_and\_save(df,path\_save\_ID).

**Argument:**

* df (dataframe)
* path\_save\_ID

**Details:**

First, we split the mother dataframe. Then each sub dataframe is saved in the same folder as the mother dataframe, using path\_save\_ID.

**Values:** no value

### **e) format\_date**

**Title:** format\_date

**Description:** Format the date.

**Usage:** format\_date(df,param).

**Argument:**

* df (dataframe)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

Depending on the original date, we change it to be in a date format for later use. The original format, informed by the user with the parameter vector, can be in a year/month/day or a day/month/year form for example and is transformed into a year-month-day form.

**Values:** df (dataframe)

### **f) real\_line\_error**

**Title:** real\_line\_error

**Description:** gives the real line of an error.

**Usage:** real\_line\_error(list\_all\_error)

**Argument:** list\_all\_error (list of 2 vectors : err\_previous, the precedent error index, and err\_new, the new error index)

**Details:**

The function is necessary because there are different errors no deleted at the same time. And so, the real line/index of an error can be false, with a shift depending on the number of line inferior to this line error deleted.

First, the 2 values of the list are separated to be easily used.

If err\_previous or err\_new are empty, the function is not necessary, so err\_new is returned.

If it’s not the case, the idea is to know how many line errors inferiors of new lines errors were deleted to know the real values.

**Values:** err\_new (the new errors index)

### **f) time\_after\_midnight**

**Title:** time\_after\_midnight

**Description:** correct the time after midnight error (25:10:03 instead of 01:10:03)

**Usage:** time\_after\_midnight(df,choice\_save)

**Argument:**

* df (dataframe)
* choice\_save (if we want the index of midnight error or the new corrected dataframe) = “index”, “change\_df”

**Details:**

The function helps to correct time when we have midnight error (25:10:03 instead of 01:10:03 for example).

First, we detect the index with time after midnight error. I have limited it to 25, 26, 27, 28 and 29 for the aberrant hours.

Then, if we just want the index to record it (choice 1), we return the index. If we want to change the dataframe (choice 2), for each error, I replace the hour by the correct one.

**Values:**

* idx\_error\_after\_midnight (index of midnight error) if choice\_save = “index”
* df (dataframe) if choice\_save = “change\_df”

### **g) error\_after\_midnight**

**Title:** error\_after\_midnight

**Description:** notes midnight errors in the error file

**Usage:** error\_after\_midnight(df\_error,idx)

**Argument:**

* df\_error (dataframe of errors recorded)
* idx (index of midnight errors)

**Details:**

First, we add a title to separate the different types of error. If I have an error, I note it in the txt file. If not, I note the fact that there is no midnight error.

**Values:**

* df\_error (dataframe of errors recorded)

### **h) format\_time**

**Title:** format\_time

**Description:** convert time in correct format

**Usage:** error\_after\_midnight(df,df\_error,idx)

**Argument:**

* df (dataframe)

**Details:**

We convert time column in 3 steps, following the “%H:%M:%S” (hour:minute:second) format.

**Values:**

* df (dataframe)

### **i) replace\_or\_delete\_data**

**Title:** replace\_or\_delete\_data

**Description:** for each error, permit to delete it, replace it manually or replace it automatically

**Usage:** replace\_or\_delete\_data(df,idx,category)

**Argument:**

* df (dataframe)
* idx (error index)
* category (category of the error) : “lon\_lat” (longitude, latitude) or “datetime”
* choice\_repl\_del : only “delete” here. Because of the program I have decided to just delete the errors and to not propose to replace them.

**Details:**

For each error, the user has 3 choices: delete it, replace it manually or replace it automatically (not possible if it’s the first or the last line of the dataframe).

If we delete, it is added to a vector and deleted at the end of the function to not impact the real lines of errors.

If we replace manually, we ask the user to write the new value and add a 1 (longitude latitude error) or 2 (datetime error) to the quality column, as the error is kept. The quality column value is 0 by default.

If we replace automatically, we change the value by the mean of the 2 values below and above. We add also a 1 or 2 in the quality column.

**Values:**

* df (dataframe)

### **j) jet\_lag**

**Title:** jet\_lag

**Description:** if we are in a UTC (Universal Time Coordinated) format, permit to obtain the jet lag necessary to correct the datetime into HLOC (Hour LOCal) in the new\_datetime\_column function

**Usage:** jet\_lag(df)

**Argument:**

* df (dataframe)

**Details:**

If we are in a UTC (Universal Time Coordinated) format for hours, we need to correct the hours to obtain HLOC (Hours LOCal). This function permits to obtain the jet lag necessary.

First, we obtain the time zone with tz\_lookup\_coords() function, taking into account the latitude and longitude.

Second, we obtain a dataframe including the offset between the UTC and HLOC with tz\_offset() function, considering the date and the time zone. We only use the jet lag value, indicated in the utc\_offset\_h column.

**Values:**

* offset$utc\_offset\_h[1] = value of the jet lag (can be negative)

**References:**

<https://statisticsglobe.com/difftime-r-function/>

<https://cran.r-project.org/web/packages/lutz/lutz.pdf>

### **k) new\_datetime\_column**

**Title:** new\_datetime\_column

**Description:** combine date and hour to obtain the datetime column

**Usage:** new\_datetime\_column(df,param)

**Argument:**

* df (dataframe)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

We create the datetime column, combining the date and time columns. If we are in UTC, we need to adapt these 3 columns to be in HLOC for the study, using the jet\_lag function.

**Values:**

* df (dataframe)

### **l) find\_error\_datetime**

**Title:** find\_error\_datetime

**Description:** obtain a list of datetime error

**Usage:** find\_error\_datetime(choice,df)

**Argument:**

* choice (automatic or manual) = 0 or 1
* df (dataframe)

**Details:**

We find NA and same datetime errors. If the choice is not automatic, we add the logical and user errors.

For logical errors, we show the lines where the error is, and the user points the right one. Then we show the first and last line and he also checks, because they can’t be spotted with the logical error method.

For user errors, we just ask the user if he want to delete specific lines.

At the end, we add 2 to the quality column for the lines of the errors above.

**Values:**

* list(pb\_NA,pb\_logic,pb\_same,pb\_user) = list with NA error index, logical errors index, same datetime errors index, and errors index seen by the user

### **m) error\_datetime**

**Title:** error\_datetime

**Description:** notes datetime errors in the error file

**Usage:** error\_datetime(df,df\_error,idx\_error\_datetime)

**Argument:**

* df (dataframe of trajectory)
* df\_error (dataframe of errors recorded)
* idx\_error\_datetime(list of index of datetime errors from find\_error\_datetime function)

**Details:**

First, we add a title to separate the different types of error. If I have an error, I note it in the txt file. If not, I note the fact that there is no datetime error.

**Values:**

* df\_error (dataframe of errors recorded)

### **n) check\_replace\_time**

**Title:** check\_replace\_time

**Description:** take care of datetime problem thanks to the previous functions

**Usage:** check\_replace\_time (df,choice,idx\_error\_1st\_set)

**Argument:**

* df (dataframe of trajectory)
* choice (automatic or manual) = 0 or 1
* idx\_error\_1st\_set (list of index of datetime errors from find\_error\_datetime function)

**Details:**

We first delete all the NA/same/user errors. Then if the choice is manual, we search again the logic datetime error to have the right index, asking him what he want to do with these errors.

**Values:**

* df (dataframe)

### **o) radius\_earth**

**Title:** radius\_earth

**Description:** give the radius earth depending of the latitude

**Usage:** radius\_earth(lat)

**Argument:**

* lat (latitude)

**Details:**

**Values:**

* R (Earth radius)

**References:**

<https://gis.stackexchange.com/questions/242188/calculating-the-earth-radius-at-latitude>

### **p) dist\_ortho**

**Title:** dist\_ortho

**Description:** give the distance orthodromic between 2 points

**Usage:** dist\_ortho(lon1,lat1,lon2,lat2)

**Argument:**

* lon1 (longitude of 1st point)
* lat1 (latitude of 1st point)
* lon2 (longitude of 2nd point)
* lat2 (latitude of 2nd point)

**Details:**

Radius is calculated with the 1st latitude. Then a mathematical formula is used to obtain the distance.

**Values:**

* distance

### **q) step\_speed\_column**

**Title:** step\_speed\_column

**Description:** give the speed between 2 points in different times and positions

**Usage:** step\_speed\_column(df)

**Argument:**

* df (dataframe)

**Details:**

For each line (except the 1st) step speed is calculated thanks to distance and time.

**Values:**

* df (dataframe)

**References:**

<https://statisticsglobe.com/difftime-r-function/>

### **r) check\_replace\_lat\_lon**

**Title:** check\_replace\_lat\_lon

**Description:** find errors in latitude and longitude and eventually take care of it

**Usage:** check\_replace\_lat\_lon(df,choice)

**Argument:**

* df (dataframe)
* choice (automatic or manual) = 0 or 1

**Details:**

Find errors in latitude and longitude.

For NA errors, they are deleted.

For errors spotted by only speed superior to 50m/s, they are marked by a 3 in quality column. It’s only possible in an automatic choice. I consider if it's manual, you choose to delete or not it with the following code. It indicates the 2 wrong lines of aberrant step\_speed. In the example c is the aberrant longitude / latitude : line 3 and 4 will be spotted

a -> step\_speed = NA (line 1)

b -> step\_speed depends on a/b (line 2)

c -> step\_speed depend on b/c (line 3) = spotted

d -> step\_speed depend on c/d (line 4) = spotted

However, depending on the points, you can have only step\_speed(line3) < 50 and step\_speed(line4) > 50. Or the opposite. So, it is needed to be cautious.

For longitude latitude seen by the user, he chooses a minimum and maximum for longitude and latitude, and a maximum for step\_speed, with plots to help him.

At the end the user can choose to take care of the errors or not

**Values:**

* df (dataframe)

### **r) error\_lon\_lat**

**Title:** error\_lon\_lat

**Description:** notes longitude latitude errors in the error file

**Usage:** error\_lon\_lat(idx\_NA,idx\_error,df\_error,idx\_speed,choice,error\_before)

**Argument:**

* idx\_NA (index of NA errors)
* idx\_error (index of errors seen by the user)
* df\_error (dataframe of errors recorded)
* idx\_speed (index of speed errors seen automatically)
* error\_before (index of previous errors)

**Details:**

Write the longitude latitude errors in the error file, and find the true line for the different errors

**Values:**

* list(df\_error,(sort(unique(c(error\_before,idx\_NA,idx\_error)))),idx\_speed)) = list of the changed arguments

### **r) error\_kept**

**Title:** error\_kept

**Description:** notes errors kept in the error file

**Usage:** error\_kept(df,df\_error,choice, index\_raw\_error\_after\_datetime\_lon\_lat)

**Argument:**

* df (dataframe)
* df\_error (dataframe of errors recorded)
* choice (automatic or manual) = 0 or 1
* index\_raw\_error\_after\_datetime\_lon\_lat = index of datetime and longitude latitude errors

**Details:**

Write the longitude latitude errors in the error file, and find the true line for the different errors. For datetime errors kept, they are spotted by a value of 2 in the quality column. Then they are written in the error file. Same for longitude latitude errors kept (value of 1) and speed errors kept (value of 3). The last one is possible only in an automatic choice, because if it’s manual the user choose what he want to delete and it is in the longitude latitude error (value of 1).

**Values:**

* df\_error (dataframe of error)

### **s) cap**

**Title:** cap

**Description:** give the cap (direction) between 2 points

**Usage:** cap(lon1,lat1,lon2,lat2)

**Argument:**

* lon1 (longitude of 1st point)
* lat1 (latitude of 1st point)
* lon2 (longitude of 2nd point)
* lat2 (latitude of 2nd point)

**Details:**

Give the cap (direction) between 2 points defined by latitude and longitude. Mathematical formula in trigonometric is used.

**Values:**

* cap\*180/pi : the cap converted

### **o) step\_direction\_bird**

**Title:** step\_direction\_bird

**Description:** give thedifference of caps

**Usage:** step\_direction\_bird(df)

**Argument:**

* df (dataframe)

**Details:**

Give the difference between 2 caps, and so the step direction thanks to trigonometry.

**Values:**

* df (dataframe)

### **p) gaps\_column**

**Title:** gaps\_column

**Description:** permit to quantify when a time difference is normal comparing to the others or unusual

**Usage:** gaps\_column (df)

**Argument:**

* df (dataframe)

**Details:**

The resolution is first calculated by the median of all the time differences. Then with the time difference (+ or -) we can have the gap comparing to the others.

**Values:**

* df (dataframe)

### **q) position\_colony**

**Title:** position\_colony

**Description:** permit to obtain the position of the colony automatically, manually or by choosing in a list

**Usage:** position\_colony(df,param)

**Argument:**

* df (dataframe)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

The 15e element of the vector param define the option chosen. If we choose the option “automatic”, it calculates the colony position by the median with the points where the bird is not moving.

If we choose the option “manually”, the user enters the coordinates.

If these 2 options aren’t chosen, that means the user has entered the number of the colony chosen in the dataframe. Then the colony position is saved.

**Values:**

* position (position of the colony)

### **r) distance\_colony**

**Title:** distance\_colony

**Description:** permit to obtain the distance between the bird and the colony

**Usage:** distance\_colony(df,param)

**Argument:**

* df (dataframe)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

Use the position\_colony function to obtain the position of the colony. Then for each position it calculates the orthodromic distance as a new column.

**Values:**

* df (dataframe)

### **s) detect\_trip**

**Title:** detect\_trip

**Description:** permit to detect when a bird is on a trip

**Usage:** detect\_trip(choice,df)

**Argument:**

* df (dataframe)
* choice (automatic or manual) = 0 or 1

**Details:**

The concept is that a bird is on a trip if he is at least 500 meters away from its colony position. And a step speed higher than 5 m/s indicates the beginning of its trip. Each trip has a specific number, beginning by 1. No trip gives a 0 value in the trip column.

First, the vectors containing the beginning idx\_trip\_start and the end idx\_trip\_end of each trip are calculated by the threshold distance, with a few exceptions.

Then, with the speed, we find the beginning of the trip and the end with a while. If one trip is mixed with another, we merge them thanks to the nb\_sub variable.

We finally add a new column to the trajectory dataframe.

**Values:**

* df (dataframe)

### **t) when\_is\_night**

**Title:** when\_is\_night

**Description:** indicate when it’s night

**Usage:** when\_is\_night(df)

**Argument:**

* df (dataframe)

**Details:**

Depending of the time zone, it permits to know when it’s night thanks to 2 values : night and nightEnd.

"night" : night starts (dark enough for astronomical observations)

"nightEnd" : night ends (morning astronomical twilight starts)

**Values:**

* c(end\_night\_calc,begin\_night\_calc) = vector with time of end and beginning of night

**References:**

<https://github.com/SWotherspoon/SGAT>

<https://gallery.htmlwidgets.org/>

<https://cran.r-project.org/web/packages/suncalc/suncalc.pdf>

<https://upload.wikimedia.org/wikipedia/commons/8/88/World_Time_Zones_Map.png>

### **u) graph1**

**Title:** graph1

**Description:** plot a trajectory graph

**Usage:** graph1(df,title\_ID,param)

**Argument:**

* df (dataframe)
* title\_ID (title of the graph, and also the unique name of the bird trajectory)
* param (vector of parameters chosen by user in set\_parameter function)

**Details:**

The graph show the trajectory of the bird when it is on trip (or all the points if the bird doesn’t do any trip) depending on latitude and longitude, with a gradient of color showing the order of the points. The points with a gap superior to 1 are surrounded by a black circle. The big brown cross is the position of the colony.

**Values:**

* g1 (the graph)

**References:**

<http://www.sthda.com/french/wiki/ggplot2-types-de-points-logiciel-r-et-visualisation-de-donnees>

<https://stackoverflow.com/questions/22915337/if-else-condition-in-ggplot-to-add-an-extra-layer>

<https://stackoverflow.com/questions/45505388/add-condition-geom-point-ggplot2-r>

### **v) graph2**

**Title:** graph2

**Description:** plots a graph showing the distance to the colony depending of datetime with an indication of the speed and the light

**Usage:** graph2(df,title\_ID)

**Argument:**

* df (dataframe)
* title\_ID (title of the graph, and also the unique name of the bird trajectory)

**Details:**

The graph shows the distance to the colony depending of datetime with an indication of the speed and the light.

First, we create a new column df$night which indicate if the point is in day or night, creating a background different.

Then we take care of x lines of different colors showing beginning an end of trips. Orange is the beginning, red the end, and yellow indicates a one-line trip.

Finally, we plot the graph, with an exception if there is no trip.

**Values:**

* g2 (the graph)

**References:**

<https://www.faqcode4u.com/faq/46141/add-legend-to-geom-vline>

### **w) graph3**

**Title:** graph3

**Description:** plots a graph showing step\_speed depending on step\_direction

**Usage:** graph3(df,title\_ID)

**Argument:**

* df (dataframe)
* title\_ID (title of the graph, and also the unique name of the bird trajectory)

**Details:**

The graph shows step\_speed depending on step\_direction. It permits to check behavioral patterns

**Values:**

* g3 (the graph)

### **x) graph4**

**Title:** graph4

**Description:** plots a graph showing a difference of time histogramm

**Usage:** graph4(df,title\_ID)

**Argument:**

* df (dataframe)
* title\_ID (title of the graph, and also the unique name of the bird trajectory)

**Details:**

The graph shows time difference by histogram. The color depend of the gap factor (“inf” if gap is <1 and “sup” if gap>1). There are 2 more dotted lines, showing the mean for each factor. The resolution (median) is a x line in black

**Values:**

* g4 (the graph)

### **y) ID\_bird**

**Title:** ID\_bird

**Description:** create 2 news columns for the unique ID of bird trajectory

**Usage:** ID\_bird(df,name)

**Argument:**

* df (dataframe)
* name (name of the file without .csv/.txt)

**Details:**

Create 2 columns : IDbird (the unique name of the bird trajectory), and trip\_ID (IDbird with number of trip).

**Values:**

* df (dataframe)

### **z) format\_data\_in\_a\_folder**

**Title:** format\_data\_in\_a\_folder

**Description:** uses a large panel of functions to format and add interesting variables to a trajectory dataset

**Usage:** format\_data\_in\_a\_folder(choice,the\_path,to\_separate\_or\_no)

**Argument:**

* choice (automatic or manual) = 0 or 1
* the\_path (path of the folder where the files are)
* to\_separate\_or\_no (indicate if we just format or separate by ID) = "no\_separate", "separate"

**Details:**

In a specific folder, all the files are treated. First, the parameters are set for all the files in this folder. Then the file is opened, and new variables are added (step\_speed, step\_direction …) and errors are spotted in a error file and eventually deleted / replaced. A final error file is created, with a formatted dataset in a csv form, and finally 4 graphs permitting to visualize the dataset formatted. If it’s done automatically, a “\_auto” word is added at the end of the names of the 3 files, if not a “\_manual” one is added.

**Values:**

* data (dataframe)

# **Formatting dataset : user guide**

1 – Open *Main\_by\_Marie.R* and *Functions\_by\_Marie.R*

2 – In *Main.R*, use the setwd() if needed and source()

3 – Use the part *I – find sub folders*

## **Step 1 : Axy-trek files no readable**

Question : do you have axy-trek files with columns separated by *;* or *,* and 1 line for 1 GPS data? If no, read this text. You will need to filter you axy-trek file.

1 – Put your files in *files to format.*

2. The R code linked is in *IV. Problem axy-trek*

3.1 - First case : you have space between columns and not always value for some columns. So, it’s not possible to read the files with *a read.csv()* and have columns separated. Go to *Case 1 : space so lost for column.*

3.2 – Use the following code. Due to the huge file, it’s better to do it one by one for the computer. For each axy-trek file (in csv) you will have in the same folder an axy-trek filtered, named with a combination of its name and a *\_filtered* added.

4.1 - Second case : you have too many GPS lines (50 GPS lines for 1 second for example). Your file has , as separator. Go to *Case 2: active/dry not ok, we have too many gps lines.*

4.2 – Use the following code for one csv file at a time. Due to the huge file, it’s better to do it one by one for the computer. You will have a csv filtered added.

5.1 – Third case : we have sep="," and normal columns. It's the ideal csv file. Go to *Case 3: , and normal columns.*

5.2 – Use the following code. It also permits to have 1 line for 1 second. You will have in the same folder an axy-trek filtered, named with a combination of its name and a *\_filtered* added.

## **Step 2 : the parameters**

You can set the parameters directly in the R code each time you want to format a group of same files. But it’s better to set parameters directly to gain time and use them as many times as you want.

1 – Open the *parameters - how to choose* it csv file. You have some examples to help you.

2 - Open the *parameters - already\_chosen* csv file. Add a line by choosing a name in the *vector\_name* column. Usually, I put a number followed by the initial of the island (like 10SB for group of parameters 10 used to format data coming from Santa Barbara). So, the only differences between 10SB, 10RDJ, 10SU and 10RE is just the localization of the colony. The file format is the same.

3 – Fill all the others column with the *parameters - how to choose* it csv file and the following table.

*Table ? : help for the parameter choices*

|  |  |  |
| --- | --- | --- |
| Parameter | Explanation | Choice |
| format | the format of your file | csv / txt |
| sep | the separateur | , / space / ; |
| dec | the decimal | . / , |
| reading\_pb | If you have a reading problem, like some extra lines at the beginning, or some lines popping out of nowhere in the middle/beginning of the file spotted by a special word like EVENT | no / extra\_lines / EVENT |
| spot\_extra\_lines | how you spot your extra lines like the EVENT exemple | EVENT / no / (other words) |
| num\_line | The number of lines you want to delete at the beginning of your file (linked to the choice *extra\_lines*) | 1 / 2 / 3 / … (other integers) |
| column\_date | The position of your date column | 1 / 2 / 3 / … (other integers)  0 if you have a datetime column |
| column\_time | The position of your time column | 1 / 2 / 3 / … (other integers)  0 if you have a datetime column |
| column\_latitude | The position of your latitude column | 1 / 2 / 3 / … (other integers) |
| lat\_N\_or\_S | If you have N or S specified. It means the latitude values don’t provide a minus if needed. | nothing / S / N |
| column\_longitude | The position of your longitude column | 1 / 2 / 3 / … (other integers) |
| long\_E\_or\_W | If you have E or W specified. It means the longitude values don’t provide a minus if needed. | nothing / E / W |
| format\_date | The format of your date.  %Y: the year (2022)  %y: the simplified year (22)  %m: the month (07)  %d: the day (21)  It works even for a datetime column. Exception : a 27-Nov-09 like in Perou 2009 Pescadores. You write dmy. | %Y/%m/%d  %d/%m/%Y  %Y-%m-%d  dmy |
| Hour | If your data are saved in LOCal Hour (HLOC) or in Universal Time Coordinated (UTC) | HLOC / UTC |
| location\_col | In the *metadata\_colony\_format* csv, indicate the line of your colony. If it’s not here, you can find it automatically or enter it manually. | automatic / manually / 1 / 2 / 3 / … (other integers) |
| column\_ID | If you need to separate your data by ID, indicate the position of your ID column. If not, write no. | no / 1 / 2 / 3 / … (other integers) |
| header | If you have a header or not | TRUE / FALSE |
| datetime\_or\_not | If you have a datetime column, indicate the position of your datetime column. If not, write no. | no / 1 / 2 / 3 / … (other integers) |

4 – It will be the basis of your formatting for later.

## **Step 3: the trajectory file**

The trajectory file needs to have a certain form, with at least at each line a gps data : date, time, latitude, longitude. Also, it needs to be 1 file for 1 trajectory of 1 bird.

### **Special case: Several ID in a same trajectory file**

Question : do you have several ID in a same trajectory file? If yes, read this text.

1 – Put your files in *files to format* in the folder *folders to format*.

2 - Activate the code of *II – separate ID (if needed)*

3 – Answer the questions for the parameters or call a set of parameters already done in the *parameters - already\_chosen* csv file.

4 – You will have the separated files in the same folder as the mother file

## **4. Step 4 : filling the metadata file and renaming the trajectory file**

### **a) Fill the metadata file**

1 – Put all your trajectory files in *files to format* folder and your metadata file in *metadata* folder.

2 – Follow the example below (or in the *how to name data* csv file) for the name of the metadata columns.

3 – Use the R code in *III-1. We fill the metadata with file names*. Don’t forget to change the name of your metadata file in the *read.csv()*. You will have the file\_original column filled by all your original files (raw names).

4 – Save it thanks to the code.

5 – Fill all the other columns of your metadata, following the table below (or in the *how to name data* csv file). Let empty the last 2 columns (*file\_renamed\_by\_Marie* and *file\_formatted\_by\_Marie*).

6 – Open again your metadata filled, changing the csv name in the read.csv().

7 – Use the R code in *III.2. We fille the metadata with 2 last columns.* Depending to the other columns, it will fill the *file\_renamed\_by\_Marie* and *file\_formatted\_by\_Marie* columns.

8 – Save the metadata finished.

*Figure ? : help to rename the data trajectory (you have it in the “how to name data” csv file)*

Here an example of a file renamed : FDN\_2022-04-27\_ME\_SDAC\_5\_U61560\_AXY\_AB35\_UTC:

* FDN is the archipelago abbreviation
* 2022-04-27 is the date
* ME is the island abbreviation
* SDAC is the species abbreviation
* 5 the identification individual (ID)
* U61560 the band
* AXY the sensor name abbreviation
* AB35 the identification GPS
* UTC the datetime parameter

*Table ? : help to fill the metadata (you have it in the “how to name data” csv file)*

|  |  |  |
| --- | --- | --- |
| Metadata column | Explanation | Help and example |
| archipelago | Name of the archipelago |  |
| archipelago\_abb | Abbreviation of the archipelago name | FDN : Fernando de Noronha  PSC: Pescadores  GNP: Guanape  SPSP: Sao Pedro e Sao Paulo  ABR: Abrolhos  RDJ: Rio de Janeiro |
| date | Date indicating the beginning of the trip | Always in the days/month/year format in the column. It is automatically formatted in year-month-day for the new name |
| island | Name of the island |  |
| Island\_abb | Abbreviation of the island name | ME: Meio (FDN)  CH: Chapeu (CH)  GS: Guanape South (GNP)  GN: Guanape North (GNP)  BE: Belmonte (SPSP)  SU: Sueste (ABR)  RE: Redonda (ABR)  SI: Siriba (ABR)  Can be NA if not precised. |
| species | Name of the species |  |
| species abbreviation | Abbreviation of the species name | SDAC : Sula dactylatra  SULA: Sula sula  SLEU: Sula leucogaster  SVAR: Sula variegata  LBOU: Leucocarbo bougainvilli  PHLE: Phaethon lepturus  PHAE: Phaethon aethereus  Can be NA if not precised. |
| bird\_id | Bird identification | 1, 2, 3 … (other integers)  Can be NA if not precised. |
| band | Bird band | U61560, …  Can be NA if not precised. |
| sex | Bird sex | M: Male  F: Female  Can be NA if not precised. |
| sensor\_name | Name of the sensor |  |
| sensor\_name\_abb | Abbreviation of the sensor name | GIP: GIPSY  IGO: i-gotU  MGL: MiniGPSlog  AXY: Axy-trek |
| gps\_id | Gps identification | AB35, … |
| local\_hour | Local hour or universal time coordinated | HLOC: LOCal Hour  UTC: Universal Time Coordinated |
| file\_original | Name of the original file |  |
| file\_renamed\_by\_Marie | New name of the original file |  |
| file\_formatted\_by\_Marie | Name of the formatted file | Can be “\_auto” or “manual” at the end of the name |

### **b) Rename the trajectory file**

1 – Open again your metadata filled, changing the csv name in the read.csv().

2 – Use the R code in *III.3. We rename the files*.

3 – All your raw files are renamed, from the *file\_original* column to the *file\_renamed\_by\_Marie* column. If it’s doesn’t work, it means you haven’t correctly written the name of the raw file in the *file\_original* column.

## **5. Step 5 : formatting the trajectory file**

After having renamed your files, you can format them.

### **a) Format automatically**

1 – You put your files in *files to format* in *folder to format.*

2 – You open the *Main.R* file. In *V. Format files*, you use the 1st paragraph where it is said *## We format the files well-named automatically*. It will use all the functions in the *Functions.R* file, lead by the main function *format\_data\_in\_a\_folder*.

3 – It will ask you if you have already a set of parameters in the *parameters - already\_chosen* csv file, or if you want to choose them now: *If your parameters are the same as one of those, write the vector\_name, if not, write 0*.

3.1 – If you have already a line in this csv with all the parameters set, just write the name of this group, already written in the *vector\_name* column.

3.2 – If it’s not the case, write 0 and follow the indication to fill the vector of parameters needed to format the file.

4 – Just wait for your files to be formatted. It will create 3 more files for each formatting: a graph file, a formatted csv, and an error file. They all have an *\_auto* at the end of the name because it was formatted automatically.

### **b) How to read the 3 files created to know if it is needed to format it manually**

You will have 3 more files created for each formatting like said before: a formatted csv, an error file and 4 graphs in 1 file. Here a guide to be able to understand them and to decide if you want to manually format the file.

**The formatted csv**

The formatted csv has several more variables than the date, time (or datetime), longitude, latitude columns of the original file. Let me explain them all to you:

* date: it is now in the year-month-day format. Can be different from the original one if it was in UTC parameter.
* time: same column. Can be different from the original one if it was in UTC parameter.
* datetime: a combination of date and time. Can be different from the original one if it was in UTC parameter.
* quality : indicate the quality of the values of the line. 0 by default. If it’s 1, it indicates a bad quality in longitude / latitude. If it’s 2, it indicates a bad quality in datetime. If it’s 3, in indicates a bad quality by speed calculated.
* step\_speed\_corrected : the new step speed calculated if errors, and so lines, were deleted.
* step\_speed: the speed between 2 points.
* diff\_time: the difference of time between 2 points.
* step\_direction: the difference of direction between 3 points.
* gaps: the gap between the diff\_time between 2 points, and the median of all diff\_time. Permits to assess the quality.
* dist\_colony: the distance between the bird and the colony in a point
* trip: indicate if the bird is not in a trip (value of 0), or if he is. In that case, it indicates the number of the trip (1st trip, 2nd trip, ect) by a integer value (1,2, …). We consider a bird is on trip when it is 500 meters away from its colony, and we find the departure of its trip by a step speed superior to 5m/s.
* IDbird : the identification of the bird. Here the unique name, or more precisely, the name of the file.
* Trip\_ID: a combination between the variables *trip* and *IDbird*.

**The error file**

This error file permit to check when you have error. In the automatic choice, it can show you some automatic errors. When you use the manual option, you can find more errors. Here I will explain you all the errors that can be indicated in this file. They are divided in 4 main groups: the midnight errors, the datetime errors, the longitude latitude errors, and the errors kept. If you don’t have error in a section, it will indicate it to you. If you have an error, it indicates to you the line in the original file and the type of error. Here the different errors you can find:

* Midnight error : a time after midnight (25:10:03 instead of 01:10:03)
* Datetime error: an error in time or / and date. You can have NA errors (there is NA in the file), logic errors (the succession of datetime is not logic, like 2022-02-01 then 2014-03-01 then 2022-02-02, only in manual choice), user errors (the user choose the lines he want to delete, only in manual choice), errors of same (when you have twice or more the exact same datetime).
* Longitude latitude error: an error in longitude or latitude. It can be an NA problem, or simply an aberrant one.
* Errors kept: an error kept automatically or manually by the user. In the automatic choice, the step\_speed errors are written here, you find them 2 by 2, as 2 points are needed to obtain a step speed. In the manual choice, it can simply indicate some errors spotted that you have chosen to keep. It is usually empty in the manual choice because it was chosen to simply delete the errors spotted by the user.

**The graph file**

This file contains 4 graphs to help you to visualize if there are errors or not. Here the indications to help you understand them better.  
The first graph : latitude and longitude:

* The title is the unique name of the bird trajectory.
* You have the position of the bird depending on latitude and longitude.
* The points are colored only if it is in a trip, from yellow to red passing by orange, indicating the order.
* The black circle on a point indicate that the gap is superior to 1.
* The brown cross indicates the colony position.

The second graph: distance of the colony by time and light:

* The title is the unique name of the bird trajectory.
* You have the distance between the bird and its colony.
* The points are colored depending on the step speed, from light green to dark green.
* The black circle on a point indicate that the gap is superior to 1.
* Also, you have an indication of datetime, usually every 4 hours. You can change it in “scale\_x\_datetime(date\_breaks = "4 hours",date\_labels = "%Y/%m/%d %H:%M")+ # (every hour)”, line 1692.
* Finally at the back, it it a light gray if it’s the day, or a dark gray if it’s the night. It can be not adapted if there is no point.

The third graph: step speed by step direction

* The title is the unique name of the bird trajectory.
* Here you have step speed depending on step direction. We normally can see the different behavior of the bird on the graph.
* The red circle on a point indicate that the gap is superior to 1.

The fourth and last graph: difference of time:

* The black circle on a point indicate that the gap is superior to 1.
* Here histograms showing the number of points depending on the time difference. The red histograms have a gap inferior to 1, and the blue histograms a gap superior to 1.
* The colored dotted lines indicate the mean of the group (either with a gap superior to 1 or inferior to 1).
* The black line indicates the median of all the dataset.

### **c) Format manually**

Thanks to the 4 graphs and the error file, you can consider if your dataset is not cleaned enough with the automatic formatting and so needs to be manually formatted. If you want to format it manually, here the steps.

1 – You put your files in *files to format* in *folder to format.*

2 – You open the *Main.R* file. In *V. Format files*, you use the 1st paragraph where it is said *## We format the files well-named manually*. It will use all the functions in the *Functions.R* file, lead by the main function *format\_data\_in\_a\_folder*.

3 – It will ask you if you have already a set of parameters in the *parameters - already\_chosen* csv file, or if you want to choose them now: *If your parameters are the same as one of those, write the vector\_name, if not, write 0*.

3.1 – If you have already a line in this csv with all the parameters set, just write the name of this group, already written in the *vector\_name* column.

3.2 – If it’s not the case, write 0 and follow the indication to fill the vector of parameters needed to format the file.

4 – You file will be formatted; you will have several questions to help the manual formatting.

5 – Here the datetime errors.

5.1 - You will have a datetime graph to help you.

5.2 - *Write something to see the date plot -> good way to see where the error comes from*. Write something random.

5.3 – You will have a date graph to help you.

5.4 – If you have logic errors, it will show you the 2 lines above and below the line where the program asks you if there is an error. *You will so have 5 lines. Write here the line in the 3 proposed with a problem, or 'no' if there is no problem*. Write *no* or the line of the datetime error. You will have that question as many times as you have logic errors spotted.

5.5 – Then it will show you the first two lines and the two last lines. *Write here 1 or length(df$datetime) if there is a problem, else write 'nothing'.* If you see there is a problem, write the line number, else write *nothing*.

5.6 – If you have seen a huge group of lines with an error, the program will propose you to delete them. *Write here if you want to delete specific lines (yes/no)*. Write yes if you want to delete some, or no. Then it will ask you the first and the last line of the group you want to delete : *Write here 1st index to delete then* and *Write here last index to delete*. Either you already know which line it is, either you quickly check your original csv file to write the line.

5.7 – It will ask the same lines as before. The first one was to write the correct error line. After this task, the computer will delete the automatic errors (NA and same errors). This part is to delete the right lines with an index that is maybe not the same as before. Do the same for the logic part, be cautious of the possible different index : the green line *cautious : middle line = 349 (here 349)* is here to indicate you the right line if needed.

5.8 – Cautious : write always no to the second *Write here if you want to delete specific lines (yes/no)*. This is the second time, and the first time has already permitted you to delete and write the error index.

6 – Here the longitude latitude errors.

6.1 – You will have a longitude latitude graph to help you.

6.2 – You will have the message *Here the lat/lon plot, write something to see the plot -> good way to see where the error comes from.* Write something random.

6.3 - *Do you have errors in lat lon? (yes/no).* Write yes if you have error, else write no. If you have written no, you won’t have anymore question about longitude latitude errors.

6.4 – If you have written yes, it will ask you to choose the minimum and maximum for the latitude that you want to keep. A graph is here to help you. *Here the latitude you have, which minimum do you want to put? Which maximum do you want to put?* Same for the longitude : *Here the longitude you have, which minimum do you want to put? Which maximum do you want to put?*

6.5. Then fi you have written yes, it will ask you the maximum speed you want to put with a graph to help. *Here the step\_speed you have, which limit do you want to put?*

6.6. Finally, the program asks you if you are ok with the errors you have chosen. *Here the error index you have, do you agree with it? (yes/no).* Also if you want to keep it. If no, it will delete it. *Do you want to keep the errors or not? (yes/no)*

7 – The last step you need to choose is two last parameters. It will ask you the minimum distance to consider a trip is on bird and the minimum speed linked. *In which minimum distance do you consider the bird is not anymore in the colony? In which minimum speed do you consider the bird has finished its trip?* We usually put 500m and 5m/s.

8 - It will create 3 more files for each formatting: a graph file, a formatted csv, and an error file. They all have an *\_manual* at the end of the name because it was formatted automatically.

# **Shiny App**

## **1. Presentation**

A Shiny App was also done to visualize bird trajectories dataset depending on characteristics. The level of choices for each characteristic is adapted to the dataset. It implies that the dataset can evolve, and the Shiny App will continue to fully work.

The different choices to reduce the dataset are the archipelago, the species, the sex, the band, the sensor name, and the date. Finally, the source of the color of the trajectory is chosen by the user : it can depend on the bird / unique name (one color for one bird trajectory), the species, the sex, the sensor, or the year.

After the choice, the Shiny app permit to see the table with all the data selected at the bottom. And also, the map with all the trajectories, the colonies being showed with a house icon. It is possible to zoom the map, and to click on a trajectory to let it appear the unique name.

Une image contenant carte

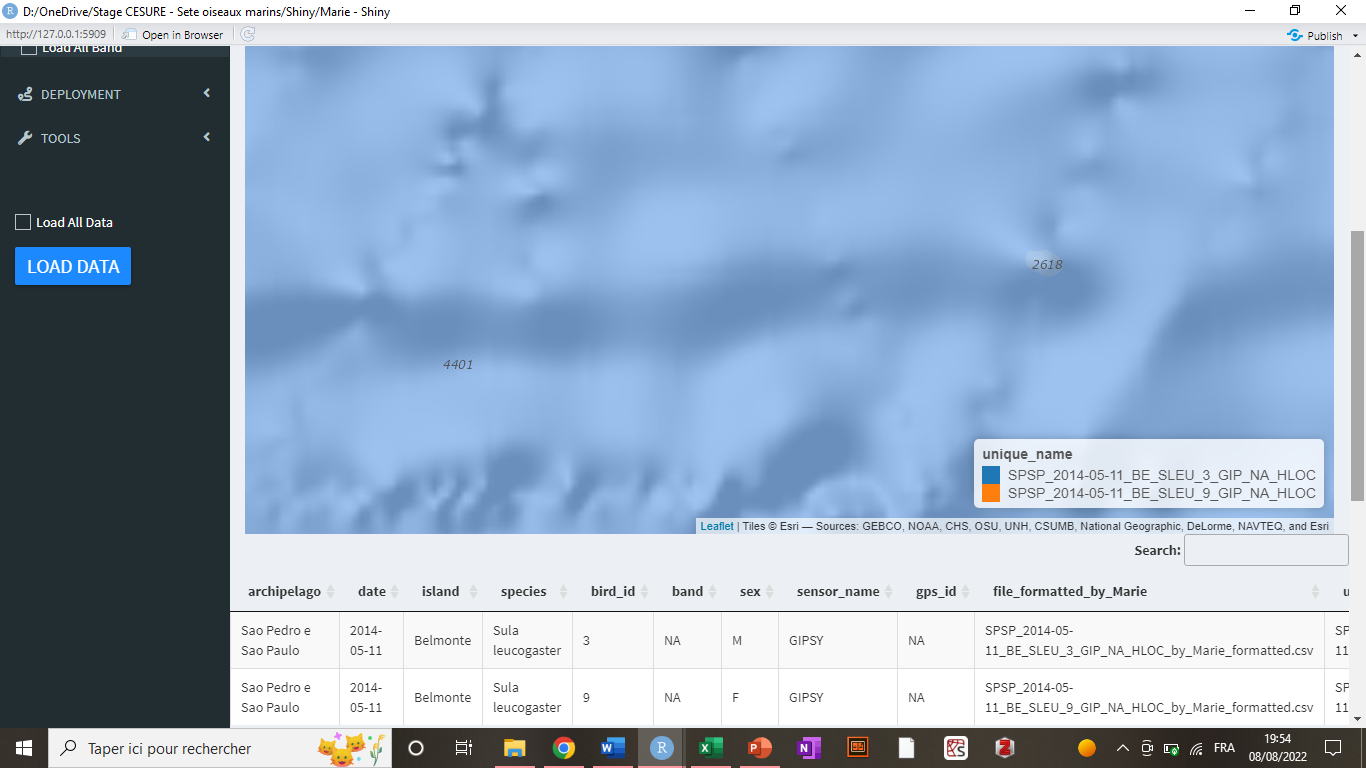
Description générée automatiquement

*Figure ? : Shiny App before loading data*

Une image contenant texte, capture d’écran, moniteur, afficher

Description générée automatiquement

*Figure ? : Shiny App after loading data (part 1)*



*Figure ? : Shiny App before loading data (part 2)*

## **2. Guide**

1 – Check in the “metadata\_colony\_shiny” if you have all the colonies.

2 – Put all your trajectories formatted in the folder "data". They need to be in a csv format. Be sure to follow the examples already done for the names of the columns.

3 - Put all your metadata in the folder "metadata". Be sure to follow the metadata examples to have the same name for the columns and the values. Example of errors : “sula dactylatra” instead of “Sula dactylatra”.

4 - These 2 folders are read by the Shiny App. The folder "metadata guilherme"/"data guilherme" and “metadata sophie"/"data sophie" are here if you want to read just 1 of the 2 datasets. In that case you need to delete the data in "data" and "metadata" and copy the group of data you want inside.

5 – Use the Shiny\_by\_Marie R code. You will have choices to filter your dataset: the archipelago, the species, the sex, the band, the sensor name, and the date. Also, you have the choice for the color : the bird / unique name (one color for one bird trajectory), the species, the sex, the sensor, or the year.

6 - Push the *Load data* button and wait for the trajectories to be loaded. It can be a little bit long; you have a window to indicate where you are in the loading. You will have at the bottom the metadata of the trajectories chosen, and a legend at the bottom right of the map, except if you have chosen the *bird* option for the colors. You can click on a trajectory in the map to have the unique\_name and identify the data and zoom if needed. The black houses are the location of the colonies by island.

# **Conclusion**

Here a program to format trajectory dataset. It was made to be use again later, thanks to the guide and the easy way to use it. It permits to format more than 500 trajectory dataset from Sophie Bertrand and Guilherme Tavares Nunes dataset.

The Shiny app is also very effective to visualize a portion of this huge dataset and is also adapted to future data.

# **Bibliography**

UNITED NATIONS (2011). Law of the Sea. Part V – Exclusive Economic Zone, article 56.

<https://en.wikipedia.org/wiki/Exclusive_economic_zone#cite_note-1>

CARLOS C. J. (2009) Seabird Diversity on Brazil: A Review. Sea Swallow 58: 17–46.

CROXALL, J. P., BUTCHART, S. H. M., LASCELLES, B., STATTERSFIELD, A. J., SULLIVAN, B., SYMES, A., & TAYLOR, P. (2012). Seabird conservation status, threats and priority actions: a global assessment. *Bird Conservation International*, *22*(1), 1–34. <https://doi.org/10.1017/s0959270912000020>

GREMILLET, D. and BOULINIER, T. (2009) Spatial ecology and conservation of seabirds facing global climate change a review. Marine Ecology Progress Series, 391, 121-137.doi10.3354/meps08212 - References - Scientific Research Publishing. (2013). Retrieved August 8, 2022, from Scirp.org website: <https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=804875>

TAVARES NUNES G., BERTRAND S. (2019) Call for proposals, application form. IRD.