Data analyses

Use of mapping tools for images annotation data

Sébastien Rochette, ThinkR

ThinkR

Contents

1. Prerequisites

1.1. Install local package {deeptools}

```
remotes::install_local("deeptools_0.0.2.tar.gz")
```

2. General exploration of the data

2.1. Packages

```
library(dplyr)
library(lubridate)
library(tidyr)
library(ggplot2)
# devtools::install_github("r-spatial/sf")
library(sf)
library(raster)
library(fasterize)
library(igraph)
library(rasterVis)
library(cowplot)
library(deeptools)
# devtools::document()
# devtools::document()
# devtools::load_all(here::here(""))
```

2.2. Colours

```
blue <- "#093564"
yellow <- "#efcc26"
grey <- "#675546"
```

2.3. Load data

```
# load data
export_file <- system.file("data_orig/export_last.csv", package = "deeptools")
liste_photo <- system.file("data_orig/liste_photo.txt", package = "deeptools")
```

2.4. Prepare data

- Cleaning of species names to be easily usable
- Add user_id combining username and date of image analysis just in case a user sees the same image two times.

```
mission2 <- readr::read_csv(export_file) %>%
  dplyr::select(-comment) %>%
  tidyr::extract(name,
          into = "datetime", regex = "_([[:digit:]]+).",
          remove = FALSE
  ) %>%
  mutate(datetime = ymd_hms(datetime, tz = "UTC")) %>%
  # clean names of species
  mutate(name_fr_clean = thinkr::clean_vec(name_fr, unique = FALSE)) %>%
  group_by(username) %>%
  mutate(
    user_id = paste(username, as.character(as.numeric(as.factor(datDeb))), sep = "-")
  ) %>%
  ungroup()
```

```
## Parsed with column specification:
## cols(
```

```
##   id = col_integer(),
##   image_id = col_integer(),
##   name = col_character(),
##   username = col_character(),
##   userlevel = col_integer(),
##   comment = col_character(),
##   datDeb = col_datetime(format = ""),
##   datFin = col_datetime(format = ""),
##   obs_code = col_character(),
##   name_fr = col_character(),
##   pos1x = col_integer(),
##   pos1y = col_integer(),
##   pos2x = col_integer(),
##   pos2y = col_integer(),
##   length = col_integer(),
##   middle_x = col_integer(),
##   middle_y = col_integer(),
##   polygon_values = col_character()
## )
```

```
## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)
```

```
## Warning: 305088 parsing failures.
## row # A tibble: 5 x 5 col      row col     expected   actual file
## ... ................. ... .........................................................
## See problems(...) for more details.
```

```
# Separate observatory dataset
mission2_MAR <- mission2 %>% filter(obs_code == "MAR")
mission2_ONC <- mission2 %>% filter(obs_code == "JDF")
```

2.5. percentage participation (part)

```
part <- mission2 %>%
  count(username) %>%
  mutate(perc = n * 100 / sum(n)) %>%
  rlang::set_names(c("UserID", "nb_annotations", "percentage")) %>%
  arrange(desc(nb_annotations)) %>%
  mutate(sumcum = cumsum(percentage))

# 463 participants
part
```

```
## # A tibble: 463 x 4
##    UserID        nb_annotations percentage sumcum
##    <chr>                  <int>      <dbl>  <dbl>
##  1 chipiok                81521      51.4    51.4
##  2 grillus33              10952       6.90   58.3
##  3 Pierre                  5250       3.31   61.6
##  4 classe                  3769       2.38   64.0
##  5 fetescience             3334       2.10   66.1
##  6 tiffk67                 3231       2.04   68.1
##  7 Audrette                2484       1.57   69.7
##  8 Kazu                    2211       1.39   71.1
##  9 macrobrachium           2098       1.32   72.4
## 10 Steatoda                1699       1.07   73.4
## # ... with 453 more rows
```

2.6. Number of participants per number of image annotated

```
nb_annot_part <- part %>%
  group_by(nb_annotations) %>%
  summarize(Nb_participants = n()) %>%
  arrange(desc(nb_annotations)) %>%
  mutate(sumcum = cumsum(Nb_participants))

nb_annot_part
```

```
## # A tibble: 170 x 3
##    nb_annotations Nb_participants sumcum
##             <int>           <int>  <int>
##  1          81521               1      1
##  2          10952               1      2
##  3           5250               1      3
##  4           3769               1      4
##  5           3334               1      5
##  6           3231               1      6
##  7           2484               1      7
##  8           2211               1      8
##  9           2098               1      9
## 10           1699               1     10
## # ... with 160 more rows
```

```
# Without Chipiok (58552 annotations)
part2 <- part[-1, 1:3]
part2$sumcum <- cumsum(part2$percentage)
```

2.7. Number of images annotated by participant

```
## Number of images annotated by participant##
nb_image_part <- mission2 %>%
  dplyr::select(image_id, username) %>%
  distinct() %>%
  count(username) %>%
  arrange(desc(n))

nb_image_part
```

```
## # A tibble: 463 x 2
##    username        n
##    <chr>       <int>
##  1 chipiok      2338
##  2 grillus33    1019
##  3 Audrette      570
##  4 classe        499
##  5 tiffk67       496
##  6 fetescience   409
##  7 Kazu          343
##  8 Steatoda      335
##  9 Pierre        306
## 10 Azaldar       273
## # ... with 453 more rows
```

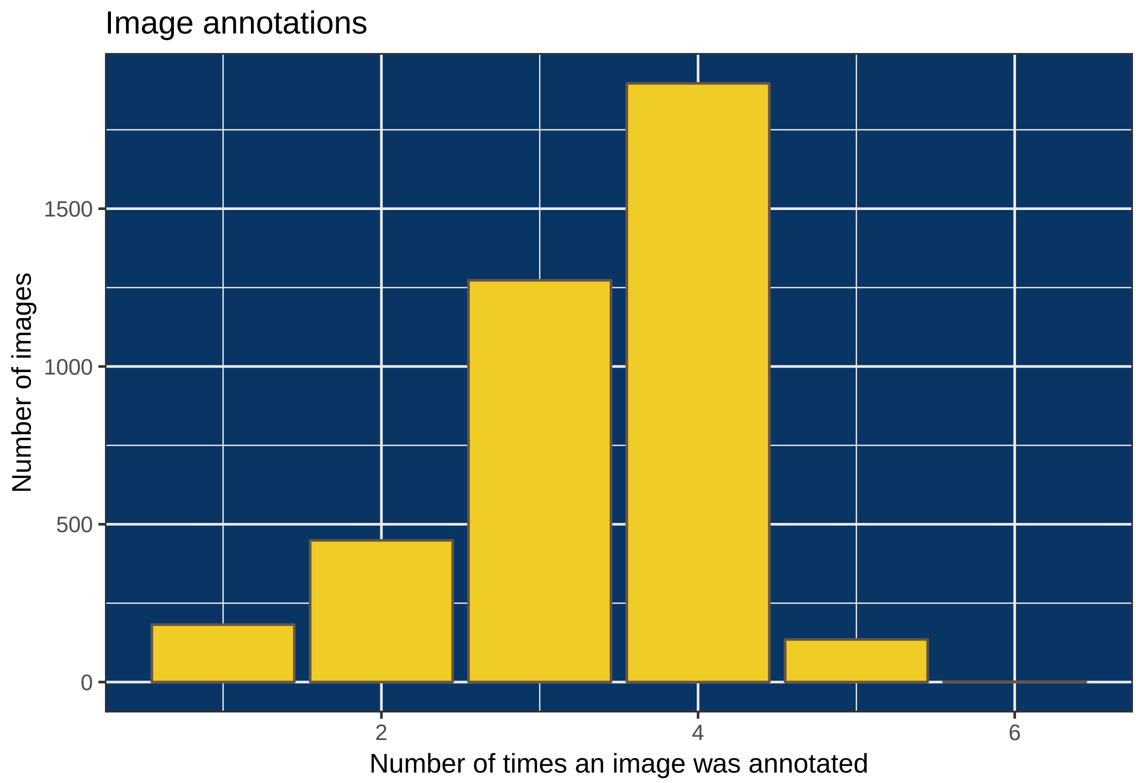
2.8. Number of time an image was annotated

```
## Number of time an image was annotated ----
nb_part_image <- mission2 %>%
  dplyr::select(image_id, username) %>%
  distinct() %>%
  dplyr::count(image_id) %>%
  arrange(desc(n))

freq_dis <- nb_part_image %>%
  count(n) %>%
  rename(n_annotation = n,
         n_images = nn)

freq_dis
```

```
## # A tibble: 6 x 2
##   n_annotation n_images
##          <int>    <int>
## 1            1      182
## 2            2      449
## 3            3     1273
## 4            4     1897
## 5            5      135
## 6            6        1
```

- histogram

```
# histogram
ggplot(nb_part_image, aes(n)) +
  geom_bar(fill = yellow, colour = grey) +
  theme_bw() +
  xlab("Number of times an image was annotated") +
  ylab("Number of images") +
  ggtitle("Image annotations") +
  theme(panel.background = element_rect(fill = blue))
```



2.9. Other tops in images in ONC

```
# Top animal
mission2_ONC %>%
  group_by(name_fr_clean) %>%
  summarise(n_images = length(unique(image_id))) %>%
  arrange(desc(n_images))
```

```
## # A tibble: 8 x 2
##   name_fr_clean            n_images
##   <chr>                       <int>
## 1 escargot_buccinide           3251
## 2 vers_polynoides              2099
## 3 crabe_araignee               2032
## 4 pycnogonide                  2028
## 5 couverture_vers_tubicole     1385
## 6 poisson_zoarcide             1361
## 7 ver_polynoide                 636
## 8 couverture_microbienne         68
```

```
# Top user
mission2_ONC %>%
  group_by(username) %>%
  summarise(n_images = length(unique(image_id))) %>%
  arrange(desc(n_images))
```

```
## # A tibble: 458 x 2
##    username    n_images
##    <chr>          <int>
##  1 chipiok         2106
##  2 grillus33        844
##  3 Audrette         516
##  4 classe           467
##  5 tiffk67          429
```

```
##   6 fetescience      370
##   7 Kazu             303
##   8 Steatoda         303
##   9 Pierre           259
##  10 Azaldar          238
## # ... with 448 more rows
```

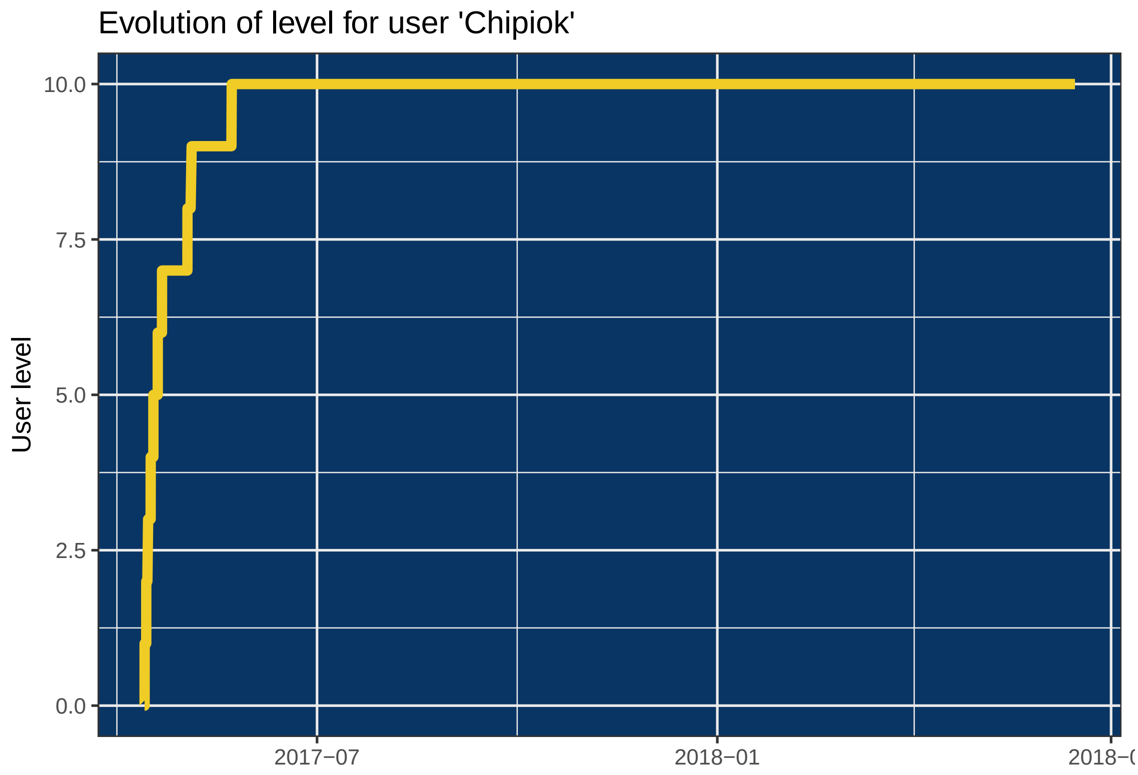
```
# Top image
mission2_ONC %>%
  group_by(image_id) %>%
  summarise(n_users = length(unique(username))) %>%
  arrange(desc(n_users))
```

```
## # A tibble: 3,392 x 2
##    image_id n_users
##       <int>   <int>
##  1    12755       6
##  2    10700       5
##  3    10705       5
##  4    10707       5
##  5    10758       5
##  6    10759       5
##  7    10829       5
##  8    10843       5
##  9    10883       5
## 10    10911       5
## # ... with 3,382 more rows
```

2.10. Exploration of userlevel in ONC

2.10.1 Userlevel: username == "chipiok"

```
mission2_ONC %>%
  filter(username == "chipiok") %>%
  group_by(datDeb) %>%
  summarize(userlevel = mean(userlevel)) %>%
  ggplot() +
  geom_line(aes(datDeb, userlevel), colour = yellow, size = 2) +
  theme_bw() +
  theme(panel.background = element_rect(fill = blue)) +
  ggtitle("Evolution of level for user 'Chipiok'") +
  ylab("User level") + xlab(NULL)
```



3. Exploration of "Buccinide" data

3.1. Packages

```
library(dplyr)
library(lubridate)
library(tidyr)
library(ggplot2)
# devtools::install_github("r-spatial/sf")
library(sf)
library(raster)
library(fasterize)
library(igraph)
library(rasterVis)
library(cowplot)
library(deeptools)
# devtools::document()
# devtools::document()
# devtools::load_all(here::here(""))
```

3.2. Colours

```
blue <- "#093564"
yellow <- "#efcc26"
grey <- "#675546"
```

3.3. Load data

```
# load data
export_file <- system.file("data_orig/export_last.csv", package = "deeptools")
liste_photo <- system.file("data_orig/liste_photo.txt", package = "deeptools")
```

3.4. Prepare data

- Cleaning of species names to be easily usable
- Add user_id combining username and date of image analysis just in case a user sees the same image two times.

```
mission2 <- readr::read_csv(export_file) %>%
  dplyr::select(-comment) %>%
  tidyr::extract(name,
          into = "datetime", regex = "_([[:digit:]]+).",
          remove = FALSE
  ) %>%
  mutate(datetime = ymd_hms(datetime, tz = "UTC")) %>%
  # clean names of species
  mutate(name_fr_clean = thinkr::clean_vec(name_fr, unique = FALSE)) %>%
  group_by(username) %>%
  mutate(
    user_id = paste(username, as.character(as.numeric(as.factor(datDeb))), sep = "-")
  ) %>%
  ungroup()
```

```
#> Parsed with column specification:
#> cols(
#>   id = col_integer(),
#>   image_id = col_integer(),
#>   name = col_character(),
#>   username = col_character(),
#>   userlevel = col_integer(),
#>   comment = col_character(),
#>   datDeb = col_datetime(format = ""),
#>   datFin = col_datetime(format = ""),
#>   obs_code = col_character(),
#>   name_fr = col_character(),
#>   pos1x = col_integer(),
#>   pos1y = col_integer(),
#>   pos2x = col_integer(),
#>   pos2y = col_integer(),
#>   length = col_integer(),
#>   middle_x = col_integer(),
#>   middle_y = col_integer(),
#>   polygon_values = col_character()
#> )

#> Warning in rbind(names(probs), probs_f): number of columns of result is not
#> a multiple of vector length (arg 1)

#> Warning: 305088 parsing failures.
#> row # A tibble: 5 x 5 col      row col     expected  actual file
#> ... ................. ... ..........................................................
#> See problems(...) for more details.
```

```
# Separate observatory dataset
mission2_MAR <- mission2 %>% filter(obs_code == "MAR")
mission2_ONC <- mission2 %>% filter(obs_code == "JDF")
```

3.5. Extraction of "Buccinide"

Function `to_carto` extract and transform data as spatial object for following analyses.

```
# Filter on Buccinide only
ONC2_bucc <- mission2_ONC %>%
  filter(name_fr_clean == "escargot_buccinide")

# Filter and transform as spatial data
ONC2_bucc_carto <- mission2_ONC %>%
  to_carto(name_fr_clean, "escargot_buccinide")
```

3.6. Exploration of annotations

3.6.1 Users

```
ONC2_bucc %>%
  count(username) %>%
  arrange(desc(n))
```

```
#> # A tibble: 453 x 2
#>    username        n
#>    <chr>       <int>
#>  1 chipiok     14183
#>  2 grillus33    5230
#>  3 tiffk67      2534
#>  4 classe       2057
#>  5 Audrette     1917
#>  6 Kazu         1819
#>  7 fetescience  1731
#>  8 Steatoda     1516
#>  9 Pierre       1367
#> 10 Azaldar      1231
#> # ... with 443 more rows
```

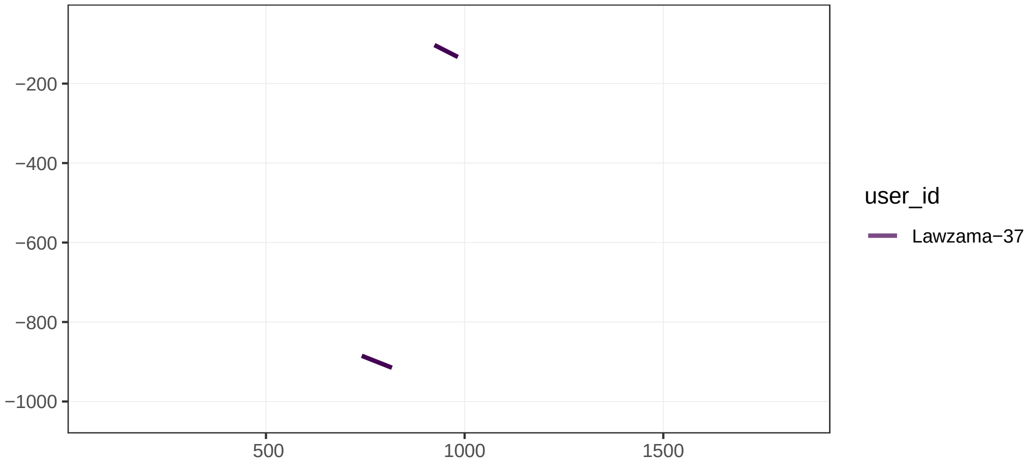
3.6.2 Images

```
ONC2_bucc %>%
  count(image_id) %>%
  arrange(desc(n))
```

```
#> # A tibble: 3,251 x 2
#>    image_id     n
#>       <int> <int>
#>  1    11873   101
#>  2    11201    88
#>  3    10747    86
#>  4    11383    86
#>  5    12794    85
#>  6    12783    83
#>  7    12200    82
#>  8    11830    80
#>  9    11379    79
#> 10    11947    78
#> # ... with 3,241 more rows
```

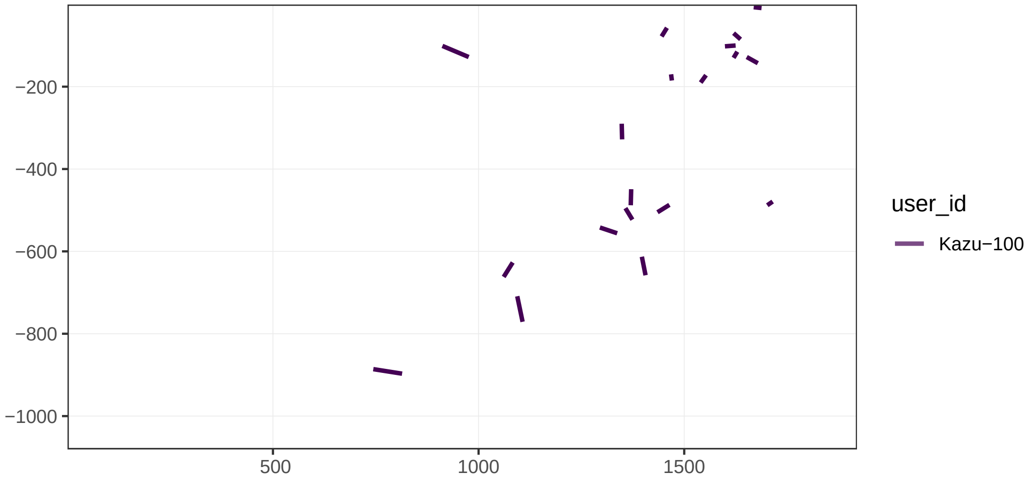
3.6.3 Example: `filter_col = username, filter_val = "Lawzama", image_id == "10681"`

```
#> # A tibble: 115 x 2
#>    image_id     n
#>       <int> <int>
#>  1    12860    12
#>  2    10957    10
#>  3    12568    10
#>  4    10853     9
#>  5    11704     8
#>  6    12350     8
#>  7    10821     7
#>  8    11383     7
#>  9    11517     7
#> 10    11705     7
#> # ... with 105 more rows
```



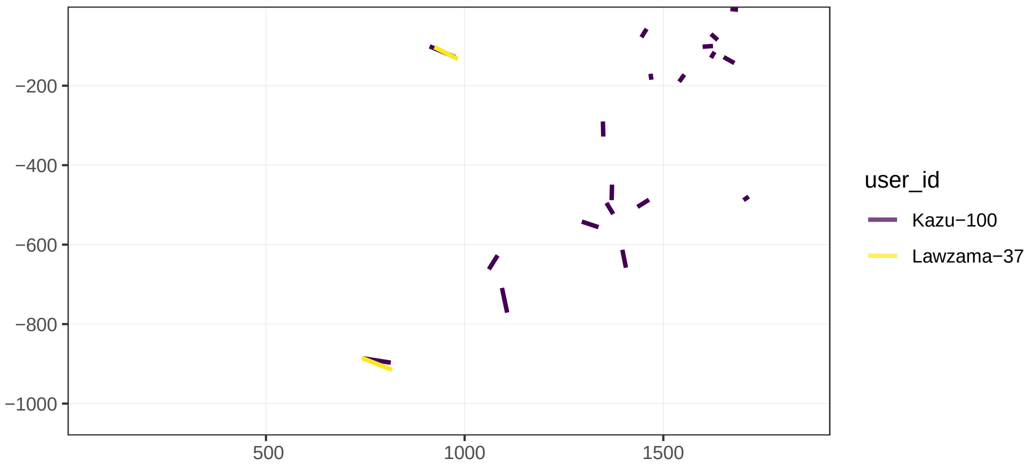
3.6.4 Example: `filter_col = username, filter_val = "Kazu", image_id == "10681"`

```r
gg_users_image(x = ONC2_bucc_carto,
               filter_col = username, filter_val = "Kazu",
               image_id = 10681)
```



3.6.5 Example: `filter_col = username, filter_val = c("Kazu", "Lawzama"), image_id == "10681"`

```
gg_users_image(x = ONC2_bucc_carto,
               filter_col = username, filter_val = c("Kazu", "Lawzama"),
               image_id = 10681)
```



3.7. Compare annotations of the same image

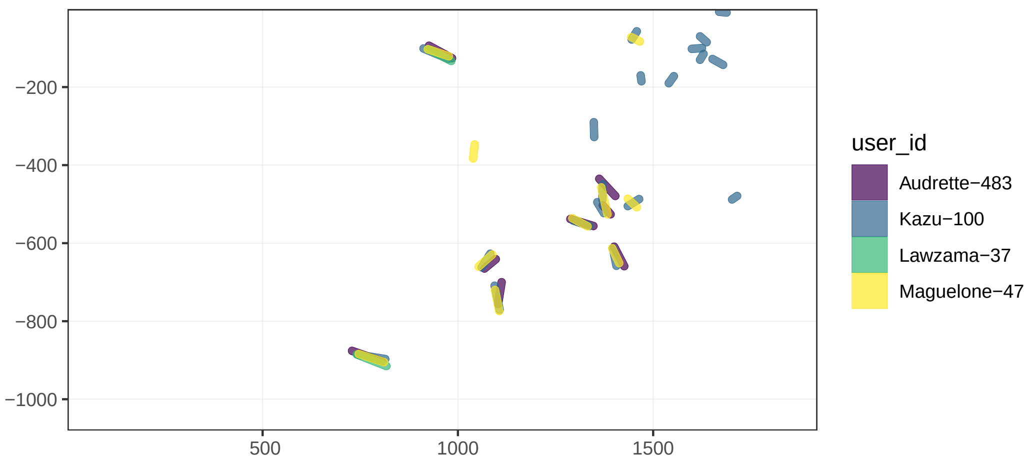
3.7.1 Super-impose annotations

This allows to check visually what sizes of buffer we can use to determine animals discovered by two persons.

```
# Number of animals seen on the image by username
ONC2_bucc %>%
  filter(image_id == 10681) %>%
  count(user_id)
```

```
#> # A tibble: 4 x 2
#>   user_id          n
#>   <chr>        <int>
#> 1 Audrette-483     8
#> 2 Kazu-100        19
#> 3 Lawzama-37       2
#> 4 Maguelone-47    10
```

```
gg_users_image(x = ONC2_bucc_carto,
               image_id = 10681, buffer = 10)
```



3.7.2 Find common animals between users

Method detailed

These steps do not have to be run as they are included in `find_groups_in_image` *function presented below. These steps are here to explain the method.*

To compute statistics on individuals on images, it is important to know if different users found the same individuals. There is a variability in positioning individuals. Hence, to find out common individuals between users, we need to look in the neighborhood. A quick exploration suggests a buffer area of 10 pixels could be enough for finding overlapping annotations.
The following steps are here to detail the complete process, but at the end, they are all included in a unique fonction `find_commons`. *Note that functions are adapted to the structure of the examle dataset, included column names.*

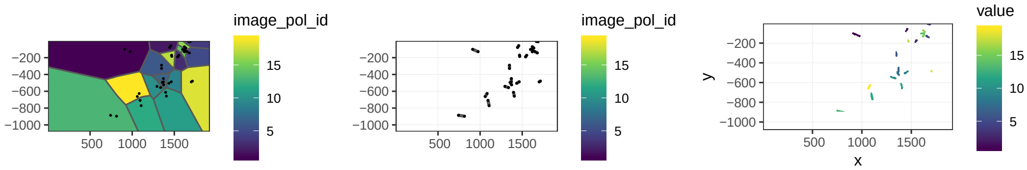
For each image steps are:

- Create Voronoi polygons around sf features by `user_id`. Voronoi is necessary to get non overlapping buffer areas of too close individuals. This also allows for super-imposed individuals.
- Crop Voronoi with buffer area. This avoids to look in a too far neighborhood.
- Transform Voronoi as raster by `user_id`. Transforming polygons into pixels allow to find out the most overlapping groups of polygons among users.
- Stack all rasters by `image_id`

```
# Choose one image for one user
image_id <- "10681"
user_id <- "Kazu-100"

bucc_image_user <- ONC2_bucc_carto %>%
  dplyr::filter(image_id == !!image_id,
                user_id == !!user_id)

r_image <- raster(bucc_image_user, res = 0.5)
dist_buffer <- 10

# Calculate voronoi for one user_id
image_user_intermediates <- voronoi_rasterize(
  x = bucc_image_user,
  dist_buffer = dist_buffer,
  r_image = r_image,
  keep_intermediate = TRUE)
```

```
#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries
```

```
# Show intermediate steps
p1 <- ggplot(image_user_intermediates$points_in_voronoi) +
  geom_sf(aes(fill = image_pol_id)) +
  geom_sf(data = st_cast(st_geometry(bucc_image_user), "POINT"), size = 0.25) +
  theme_images(x = image_user_intermediates$points_in_voronoi, fill = "c", color = NULL)

p2 <- ggplot(image_user_intermediates$points_in_voronoi_in_buffer) +
  geom_sf(aes(fill = image_pol_id)) +
    geom_sf(data = st_cast(st_geometry(bucc_image_user), "POINT"), size = 0.25) +
  theme_images(x = image_user_intermediates$points_in_voronoi, fill = "c", color = NULL)

p3 <- gplot(image_user_intermediates$voronoi_in_buffer_as_raster) +
  geom_tile(aes(fill = value)) +
  theme_images(x = image_user_intermediates$points_in_voronoi, fill = "c", color = NULL, na.

cowplot::plot_grid(plotlist = list(p1, p2, p3), ncol = 3)
```

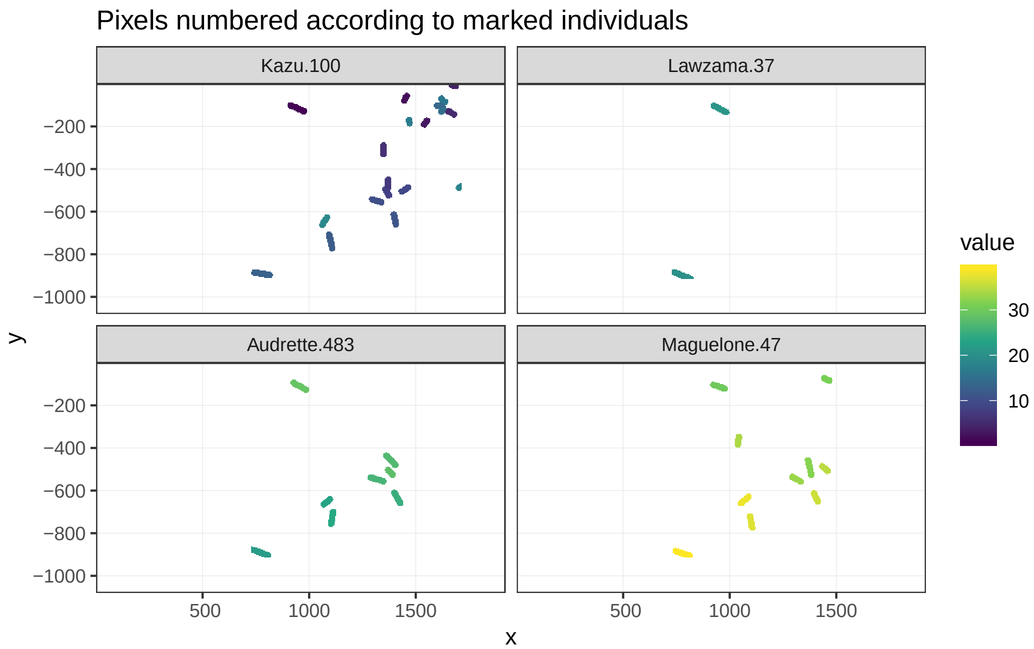


The above intermediate steps are included in the `voronoi_stacker` function, which stacks rasters of all `user_id`.

```
image_id <- "10681"
bucc_image <- ONC2_bucc_carto %>%
  dplyr::filter(image_id == !!image_id)

r_image <- raster(bucc_image, res = 0.5)
dist_buffer <- 15

bucc_voronoi_stack <- voronoi_stacker(x = bucc_image,
           dist_buffer = dist_buffer,
           r_image = r_image)
```

```
#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries
```

```
gplot(bucc_voronoi_stack) +
  geom_tile(aes(fill = value)) +
  facet_wrap(~variable) +
  theme_images(x = image_user_intermediates$points_in_voronoi,
               fill = "c", color = NULL, na.value = NA) +
  ggtitle("Pixels numbered according to marked individuals")
```



For each pixel of the stack, we can define a group of polygons with the same size as the number of `user_id`. Indeed, an important part of pixels are not covered by polygons and will be identified as `NA-...-NA-...-NA`. Once removed those pixels, we can find the most represented groups of polygons.

```
# Combine layers and find groups of polygons
bucc_groups_count <- group_pixels_count(bucc_voronoi_stack)
bucc_groups_count
```

```
#> # A tibble: 86 x 2
#>    grouped_ids n_pixels
#>    <chr>          <int>
#>  1 13-20-22-39     7614
#>  2 6-NA-NA-NA      7388
#>  3 NA-NA-NA-34     7172
#>  4 10-NA-26-33     6765
#>  5 1-21-29-30      6583
#>  6 NA-NA-27-NA     6459
#>  7 5-NA-NA-NA      6274
#>  8 3-NA-NA-NA      5492
#>  9 12-NA-24-37     5464
#> 10 14-NA-NA-NA     5186
#> # ... with 76 more rows
```

Problem is that some polygons are in more or less big groups, sometimes being in combination with two different polygons of the same user. We need to find out the best combinations of polygons to associate them to individuals really appearing on the original images.
In step 1, for each polygon independently:

- Choose group when associated to maximum other polygons
- Choose group with highest surface in common

With these two rules, some chosen groups may include polygons associated to other groups.

```
#' Find in which groups are each polygons
all_image_pol_ids <- pull(bucc_image, image_pol_id) %>% unique()

bucc_groups_top <- find_top_groups(bucc_groups_count,
                                   all_image_pol_ids)
test_ids_in_group <- test_groups_kept(bucc_groups_top)

# Show in how many groups are individuals (Should be only one)
test_ids_in_group$ids_in_groups_count
```

```
#> # A tibble: 39 x 2
#>    list_ids     n
#>       <dbl> <int>
#>  1       32     2
#>  2        1     1
#>  3        2     1
#>  4        3     1
#>  5        4     1
#>  6        5     1
#>  7        6     1
#>  8        7     1
#>  9        8     1
#> 10        9     1
#> # ... with 29 more rows
```

It is necessary to create a loop to reduce grouping possibilities based on same rules.
In step two, for each `image_pol_id` found in multiple groups, keep the only group following the above two rules. Others are removed from list of possibilities. Step one is run again without these groups. Run these two steps until each polygon is found in one and only one group. *If an `image_pol_id` has no group left, it is included in a group alone*

```r
# If image_pol_ids are not unique reduce possibilities until it is good
if (test_ids_in_group$max_groups > 1) {
  # Run again group selection while removing groups with problems
  # _Find groups to remove
  test_ids_in_group2 <- test_ids_in_group
  bucc_groups_top2 <- bucc_groups_top
  group_remove2 <- NULL

  while (test_ids_in_group2$max_groups > 1) {

    group_remove <- test_ids_in_group2$ids_in_groups_count %>%
      filter(n > 1) %>%
      inner_join(test_ids_in_group2$ids_in_groups, by = "list_ids") %>%
      inner_join(bucc_groups_top2, by = "group_kept") %>%
      dplyr::select(-image_pol_id) %>%
      distinct() %>%
      group_by(list_ids) %>%
      arrange(desc(n_pols), desc(n_pixels)) %>%
      slice(-1) %>%
      pull(group_kept) %>%
      unique()

    group_remove2 <- unique(c(group_remove2, group_remove))

    bucc_groups_top2 <- bucc_groups_count %>%
      filter(!grouped_ids %in% group_remove2) %>%
      find_top_groups(all_ids = all_image_pol_ids)

    # bucc_groups_top2 %>% filter(grepl(7, group_kept))
    # bucc_groups_top2 %>% filter(grepl(32, group_kept))

    test_ids_in_group2 <- test_groups_kept(bucc_groups_top2)

  }
  # Retrieve success grouping
  bucc_groups_top <- bucc_groups_top2
  # Last verification
  test_ids_in_group <- test_groups_kept(bucc_groups_top)
}

# Show in how many groups are individuals (Should be only one)
test_ids_in_group$ids_in_groups_count
```

```
#> # A tibble: 39 x 2
#>    list_ids     n
#>       <dbl> <int>
#>  1        1     1
#>  2        2     1
#>  3        3     1
#>  4        4     1
#>  5        5     1
#>  6        6     1
#>  7        7     1
#>  8        8     1
#>  9        9     1
#> 10       10     1
#> # ... with 29 more rows
```

Function: Find groups in one image

The above procedure is included in a unique function available for a unique image: `find_groups_in_image(x, image_id)` which adds the group name to the original dataset.
We can then verify the grouping procedure visually.

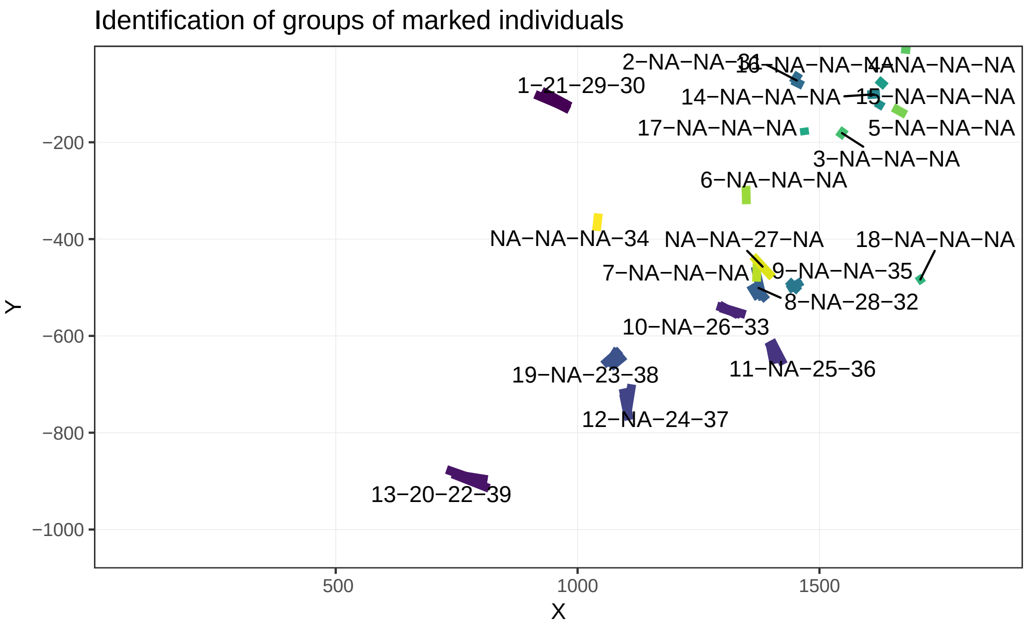
```
# Add group names in image_sf
bucc_image_grouped <- ONC2_bucc_carto %>%
  find_groups_in_image(image_id = "10681")
```

```
#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries

#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries
```

```
# Create specific image with group names
bucc_image_grouped_groups <- bucc_image_grouped %>%
  group_by(group_kept) %>%
  summarize() %>%
  st_centroid() %>%
  cbind(st_coordinates(.))
```

```
#> Warning in st_centroid.sf(.): st_centroid assumes attributes are constant
#> over geometries of x
```

```
ggplot(bucc_image_grouped %>%
         mutate(group_kept =
         forcats::fct_reorder(group_kept, desc(n_pols)))
       ) +
    geom_sf(aes(color = group_kept),
      show.legend = "line",
      size = 2 #alpha = 0.1
    ) +
  ggrepel::geom_text_repel(
    data = bucc_image_grouped_groups,
    aes(x = X, y = Y, label = group_kept)) +
    theme_images(x = bucc_image_grouped, fill = NULL, color = "d", na.value = "grey20") +
  guides(color = FALSE) +
  ggtitle("Identification of groups of marked individuals")
```



3.7.3 Find all groups for all images

Everything can be included in a unique function `find_groups_in_all_images` to explore the entire dataset at once.
This takes some time and some place:

- ~30min on dual core
- Be sure to have at least **10Go RAM** available, otherwise use `find_groups_in_image` iteratively in a loop for instance.

```
# Chunk not evaluated in Rmd as results are saved
ONC2_bucc_carto_groups <- find_groups_in_all_images(ONC2_bucc_carto, .progress = TRUE, keep_

if (!dir.exists(here::here("inst/outputs"))) {
 dir.create("inst/outputs", recursive = TRUE)
}

readr::write_rds(
  ONC2_bucc_carto_groups,
  here::here("inst/outputs", "ONC2_bucc_carto_groups.rds"),
  compress = "gz")
```

```
outwd <- system.file("outputs", package = "deeptools")
ONC2_bucc_carto_groups <- readr::read_rds(file.path(outwd, "ONC2_bucc_carto_groups.rds"))
```

3.8. Calculate statistics on images

As a reminder, a group of objects is supposed to be a unique individual.

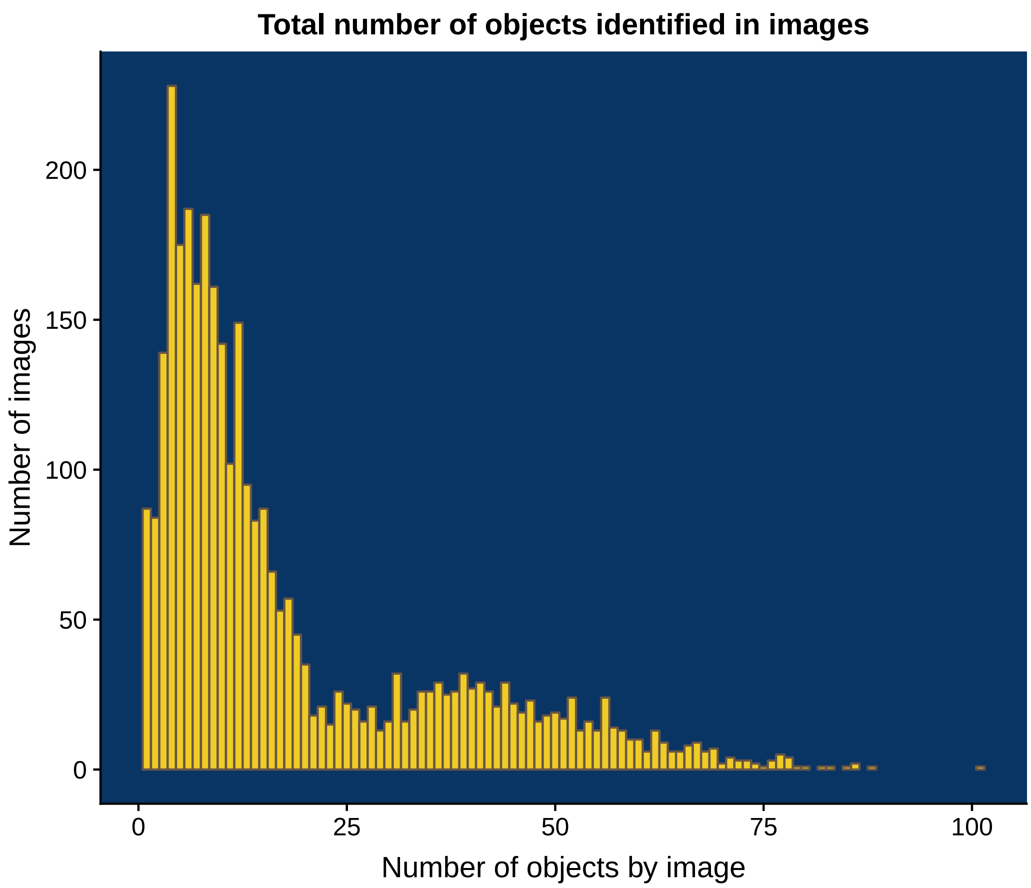
3.8.1 Number of objects per image (already known before)

```
# ONC2_bucc_carto_groups

# Number of objects per image (already known before)
bucc_nobjects <- ONC2_bucc_carto_groups %>%
  count(image_id) %>%
  rename(n_objects = n) %>%
```

```
  count(n_objects) %>%
  arrange(desc(n)) %>%
  rename(n_images = n)

# Number of marked objects by images
bucc_nobjects
```

```
#> # A tibble: 86 x 2
#>    n_objects n_images
#>        <int>    <int>
#>  1         4      228
#>  2         6      187
#>  3         8      185
#>  4         5      175
#>  5         7      162
#>  6         9      161
#>  7        12      149
#>  8        10      142
#>  9         3      139
#> 10        11      102
#> # ... with 76 more rows
```

```
# Plot
ggplot(bucc_nobjects) +
  geom_col(aes(x = n_objects, y = n_images), width = 1,
           fill = yellow, colour = grey) +
  ggtitle("Total number of objects identified in images") +
  xlab("Number of objects by image") +
  ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



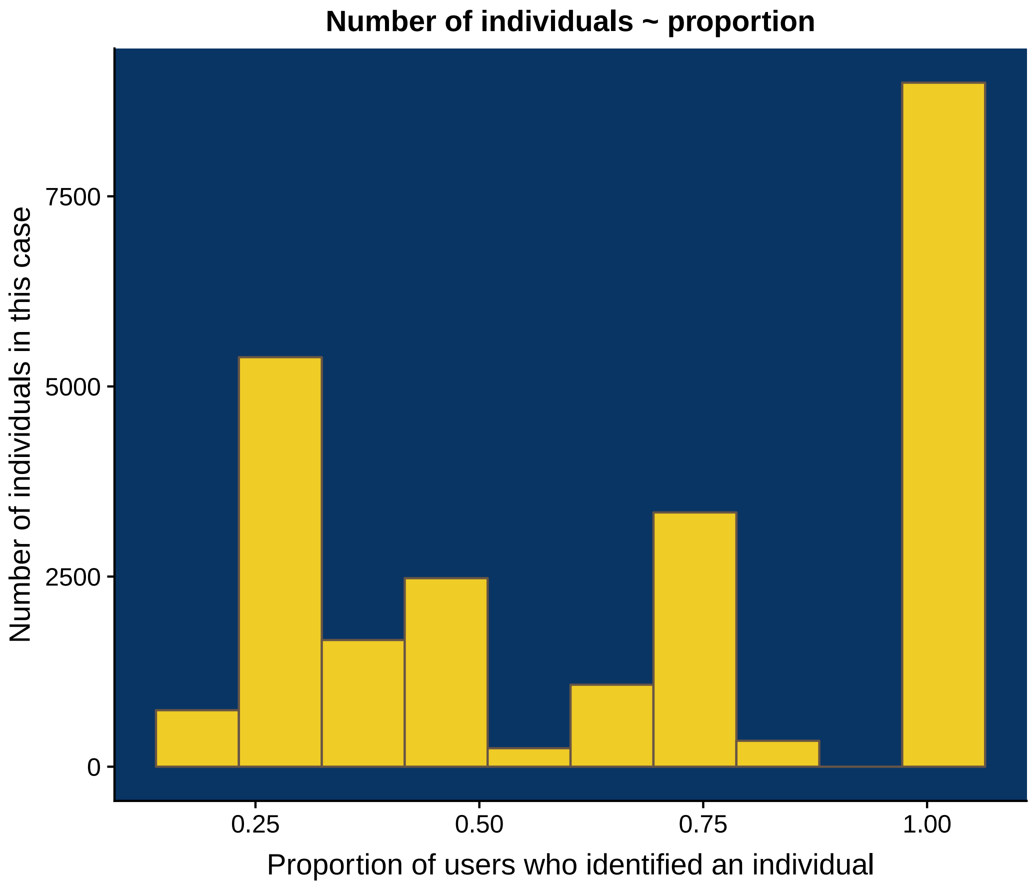
3.8.2 Statistics on groups

Calculate the number of users who marked the same individual. The `proportion` column can be used as a threshold to decide if the individual is kept. Indeed, an individual observed by only one of the users may not be a real individual.

```
# Stats on groups
bucc_groups <- ONC2_bucc_carto_groups %>%
  group_by(image_id, group_kept) %>%
  summarise(
    n_users = n(),
    n_user_id = mean(n_user_id),
    proportion = n()/mean(n_user_id)
  ) %>%
  ungroup()

# Number of individuals ~ proportion
bucc_groups
```

```
#> # A tibble: 24,272 x 5
#>    image_id group_kept  n_users n_user_id proportion
#>       <int> <chr>         <int>     <dbl>      <dbl>
#>  1    10680 1-19-NA-44        3         4       0.75
#>  2    10680 10-NA-NA-NA       1         4       0.25
#>  3    10680 11-NA-NA-NA       1         4       0.25
#>  4    10680 12-NA-NA-NA       1         4       0.25
#>  5    10680 13-26-NA-NA       2         4       0.5
#>  6    10680 14-NA-NA-NA       1         4       0.25
#>  7    10680 2-18-36-43        4         4       1
#>  8    10680 3-23-34-40        4         4       1
#>  9    10680 4-22-33-42        4         4       1
#> 10    10680 5-21-32-39        4         4       1
#> # ... with 24,262 more rows
```

```
# Plot
bucc_groups %>%
  ggplot() +
  geom_histogram(aes(proportion), bins = 10,
                 fill = yellow, colour = grey) +
  ggtitle("Number of individuals ~ proportion") +
  xlab("Proportion of users who identified an individual") +
  ylab("Number of individuals in this case") +
  theme(panel.background = element_rect(fill = blue))
```



3.8.3 Statistics on number of groups by image

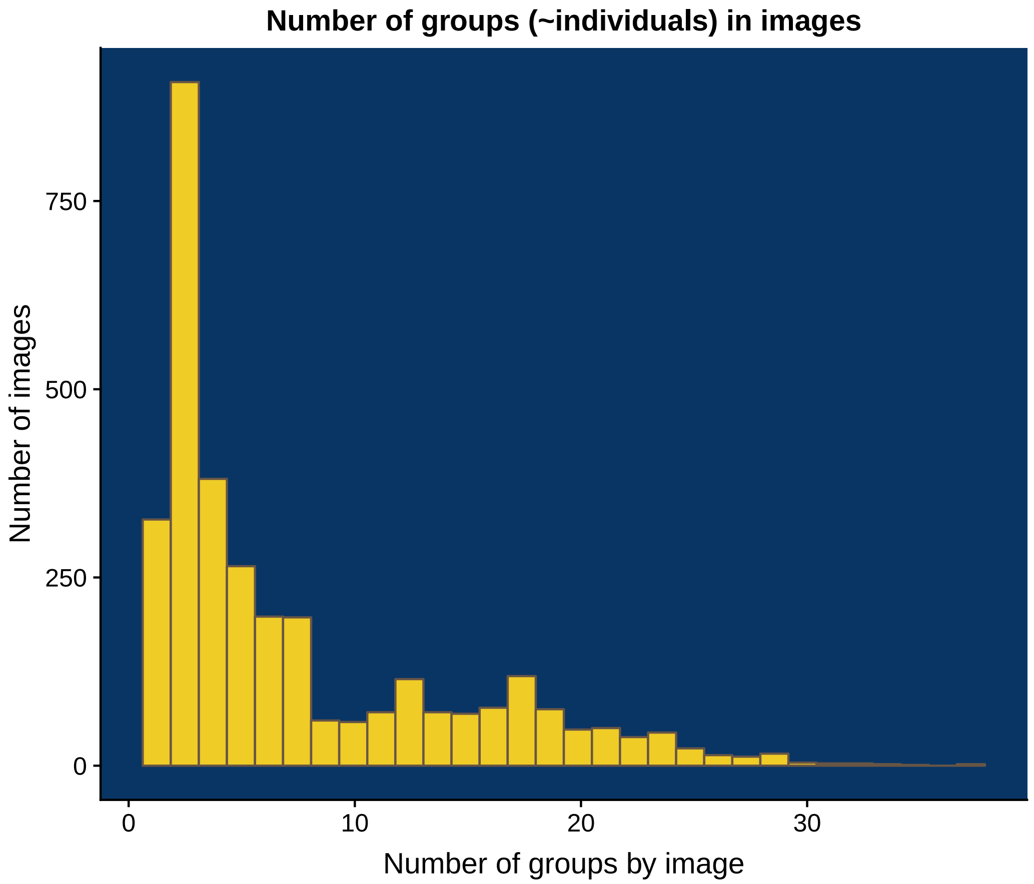
Calculate the number of groups in all images.

```
## Stats on nb groups by image
bucc_ngroups_count <- bucc_groups %>%
  group_by(image_id) %>%
  summarise(n_groups = n())

# Number of images
bucc_ngroups_count %>% count(n_groups)
```

```
#> # A tibble: 36 x 2
#>    n_groups     n
#>       <int> <int>
#>  1        1   327
#>  2        2   467
#>  3        3   441
#>  4        4   381
#>  5        5   265
#>  6        6   198
#>  7        7   120
#>  8        8    77
#>  9        9    60
#> 10       10    58
#> # ... with 26 more rows
```

```
# Plot
ggplot(bucc_ngroups_count) +
  geom_histogram(aes(x = n_groups), bins = 30,
                 fill = yellow, colour = grey) +
  ggtitle("Number of groups (~individuals) in images") +
```

```
  xlab("Number of groups by image") + ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



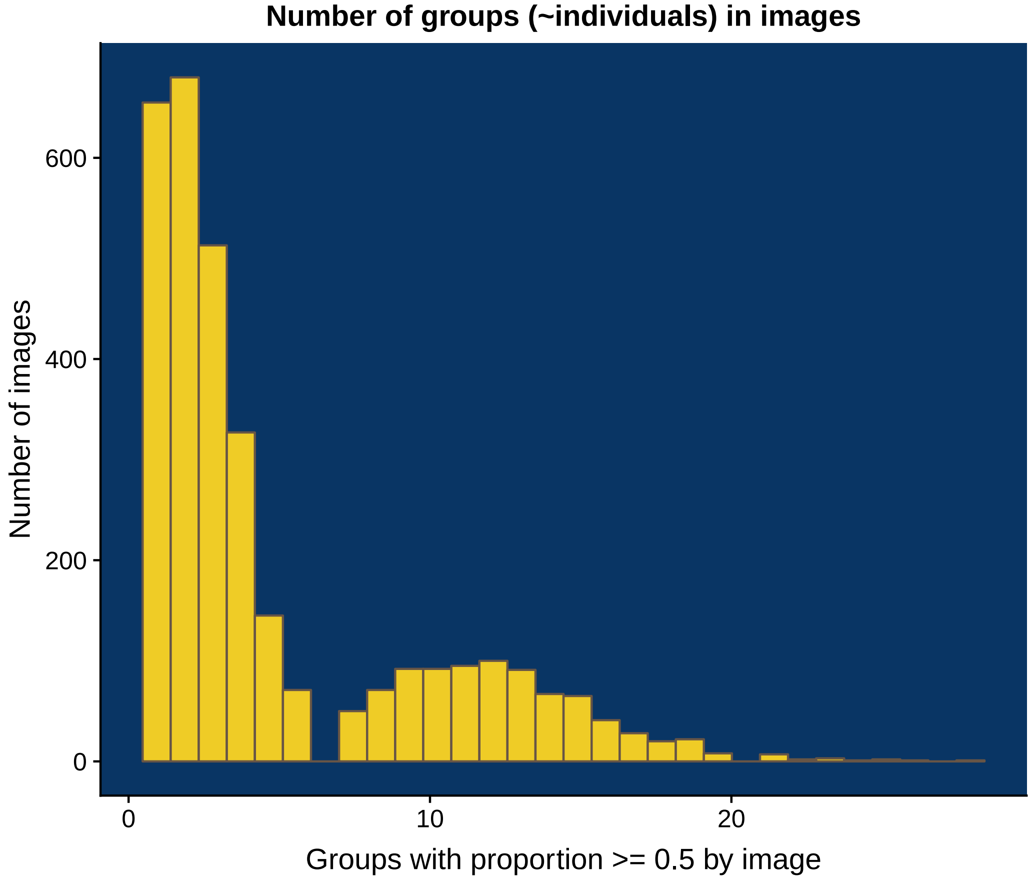
If we only keep groups identified by at least half of the users, we can recalculate the number of groups by image.

```
## Stats on nb groups by image
bucc_ngroups_count_thd <- bucc_groups %>%
  filter(proportion >= 0.5) %>%
  group_by(image_id) %>%
  summarise(n_groups = n())

# Number of images
bucc_ngroups_count_thd %>% count(n_groups)
```

```
#> # A tibble: 27 x 2
#>    n_groups     n
#>       <int> <int>
#>  1        1   655
#>  2        2   680
#>  3        3   513
#>  4        4   327
#>  5        5   145
#>  6        6    71
#>  7        7    50
#>  8        8    71
#>  9        9    92
#> 10       10    92
#> # ... with 17 more rows
```

```
# Plot
ggplot(bucc_ngroups_count_thd) +
  geom_histogram(aes(x = n_groups), bins = 30,
```

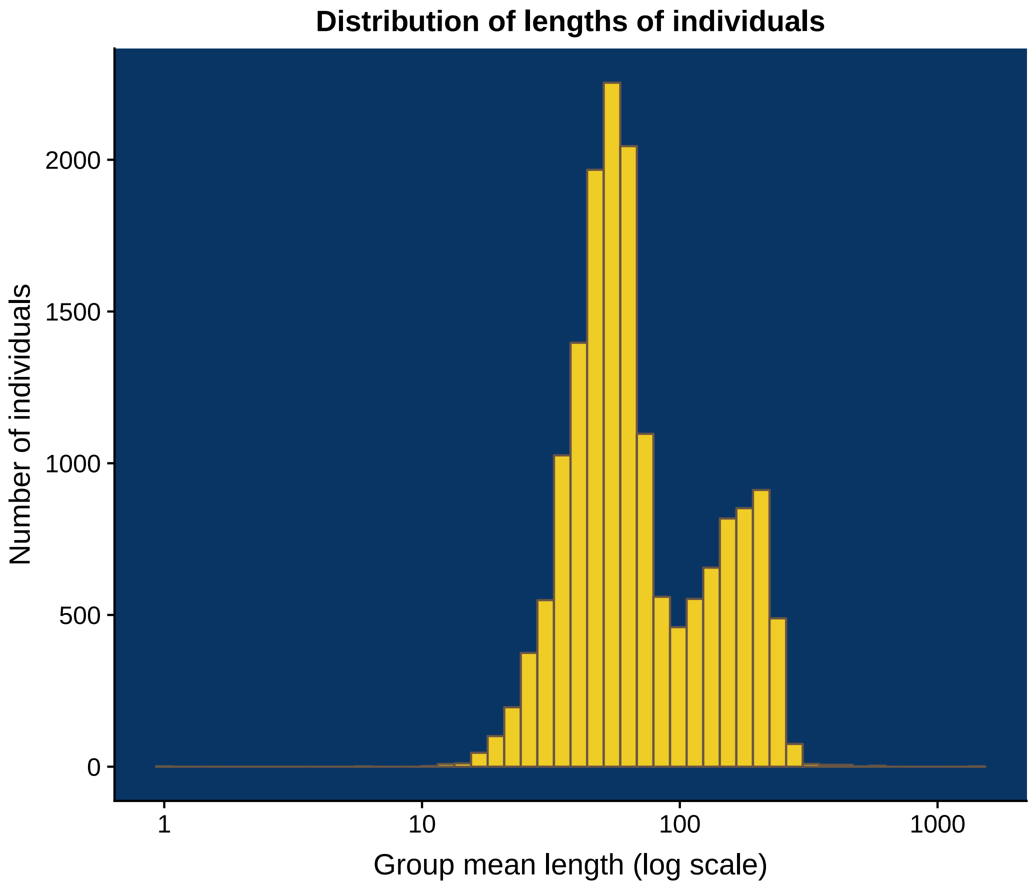
```
                  fill = yellow, colour = grey) +
  ggtitle("Number of groups (~individuals) in images") +
  xlab("Groups with proportion >= 0.5 by image") + ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



3.9. Estimate average size of individuals

We assume that a group is a unique individual.
We also remove groups identified by less than 50% of users.

```
# Mean size of individuals
bucc_lengths <- ONC2_bucc_carto_groups %>%
  left_join(bucc_groups %>%
              dplyr::select(image_id, group_kept, proportion),
            by = c("image_id", "group_kept")) %>%
  filter(proportion >= 0.5) %>%
  group_by(image_id, group_kept) %>%
  summarise(mean_length = mean(length))

ggplot(bucc_lengths) +
  geom_histogram(aes(mean_length), bins = 50,
                 fill = yellow, colour = grey) +
  scale_x_log10() +
  theme(panel.background = element_rect(fill = blue)) +
  ggtitle("Distribution of lengths of individuals") +
  xlab("Group mean length (log scale)") +
  ylab("Number of individuals")
```



4. Exploration of "Pycnogonide" data

4.1. Packages

```
library(dplyr)
library(lubridate)
library(tidyr)
library(ggplot2)
# devtools::install_github("r-spatial/sf")
library(sf)
library(raster)
library(fasterize)
library(igraph)
library(rasterVis)
library(cowplot)
library(deeptools)
# devtools::document()
# devtools::document()
# devtools::load_all(here::here(""))
```

4.2. Colours

```
blue <- "#093564"
yellow <- "#efcc26"
grey <- "#675546"
```

4.3. Load data

```
# load data
export_file <- system.file("data_orig/export_last.csv", package = "deeptools")
liste_photo <- system.file("data_orig/liste_photo.txt", package = "deeptools")
```

4.4. Prepare data

- Cleaning of species names to be easily usable
- Add user_id combining username and date of image analysis just in case a user sees the same image two times.

```
mission2 <- readr::read_csv(export_file) %>%
  dplyr::select(-comment) %>%
  tidyr::extract(name,
          into = "datetime", regex = "_([[:digit:]]+).",
          remove = FALSE
  ) %>%
  mutate(datetime = ymd_hms(datetime, tz = "UTC")) %>%
  # clean names of species
  mutate(name_fr_clean = thinkr::clean_vec(name_fr, unique = FALSE)) %>%
  group_by(username) %>%
  mutate(
    user_id = paste(username, as.character(as.numeric(as.factor(datDeb))), sep = "-")
  ) %>%
  ungroup()
```

```
#> Parsed with column specification:
#> cols(
#>   id = col_integer(),
#>   image_id = col_integer(),
#>   name = col_character(),
#>   username = col_character(),
#>   userlevel = col_integer(),
#>   comment = col_character(),
#>   datDeb = col_datetime(format = ""),
#>   datFin = col_datetime(format = ""),
#>   obs_code = col_character(),
#>   name_fr = col_character(),
#>   pos1x = col_integer(),
#>   pos1y = col_integer(),
#>   pos2x = col_integer(),
#>   pos2y = col_integer(),
#>   length = col_integer(),
#>   middle_x = col_integer(),
#>   middle_y = col_integer(),
#>   polygon_values = col_character()
#> )

#> Warning in rbind(names(probs), probs_f): number of columns of result is not
#> a multiple of vector length (arg 1)

#> Warning: 305088 parsing failures.
#> row # A tibble: 5 x 5 col      row col     expected   actual file
#> ... ................. ... ..........................................................
#> See problems(...) for more details.
```

```
# Separate observatory dataset
mission2_MAR <- mission2 %>% filter(obs_code == "MAR")
mission2_ONC <- mission2 %>% filter(obs_code == "JDF")
```

4.5. Extraction of "pycnogonide"

```
# Filter on Buccinide only
ONC2_pyc <- mission2_ONC %>%
  filter(name_fr_clean == "pycnogonide")

# Filter and transform as spatial data
ONC2_pyc_carto <- mission2_ONC %>%
  to_carto(name_fr_clean, "pycnogonide")
```

4.6. Exploration of annotations

4.6.1 Users

```
ONC2_pyc %>%
  count(username) %>%
  arrange(desc(n))
```

```
#> # A tibble: 4 x 2
#>   username        n
#>   <chr>       <int>
#> 1 chipiok     45695
#> 2 grillus33    2277
#> 3 fetescience   807
#> 4 classe        672
```

4.6.2 Images

- Number of annotations by image

```
ONC2_pyc %>%
  count(image_id) %>%
  arrange(desc(n))
```

```
#> # A tibble: 2,028 x 2
#>    image_id     n
#>       <int> <int>
#>  1    13855   180
#>  2    11725   167
#>  3    13059   165
#>  4    13373   163
#>  5    12571   158
#>  6    13510   150
#>  7    12933   146
#>  8    13578   145
#>  9    12266   135
#> 10    13405   130
#> # ... with 2,018 more rows
```

- Number of users by image

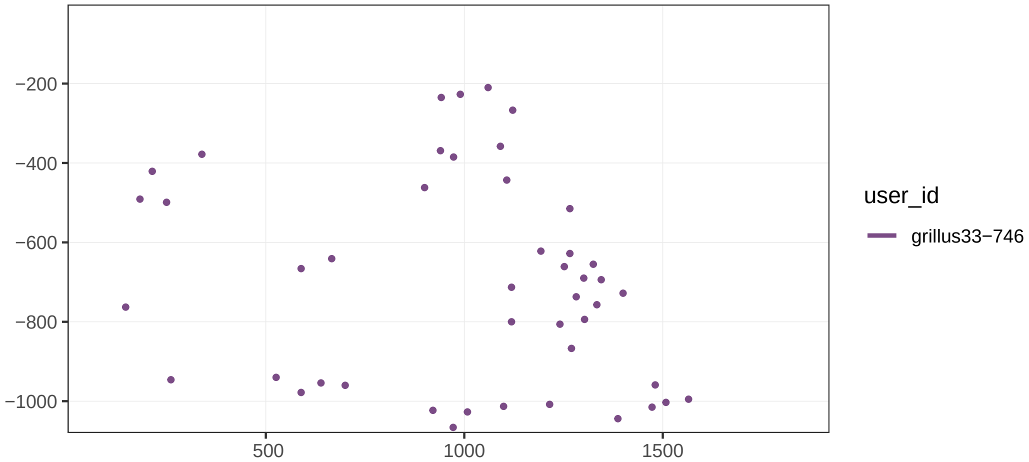
```
ONC2_pyc %>%
  group_by(image_id) %>%
  summarize(n_users = length(unique(user_id))) %>%
  arrange(desc(n_users))
```

```
#> # A tibble: 2,028 x 2
#>    image_id n_users
#>       <int>   <int>
```

```
#>  1     11425       4
#>  2     12838       4
#>  3     13988       4
#>  4     10725       3
#>  5     10784       3
#>  6     10785       3
#>  7     10874       3
#>  8     11062       3
#>  9     11083       3
#> 10     11095       3
#> # ... with 2,018 more rows
```

4.6.3 Example: `filter_col = username, filter_val = "grillus33", image_id == "10681"`

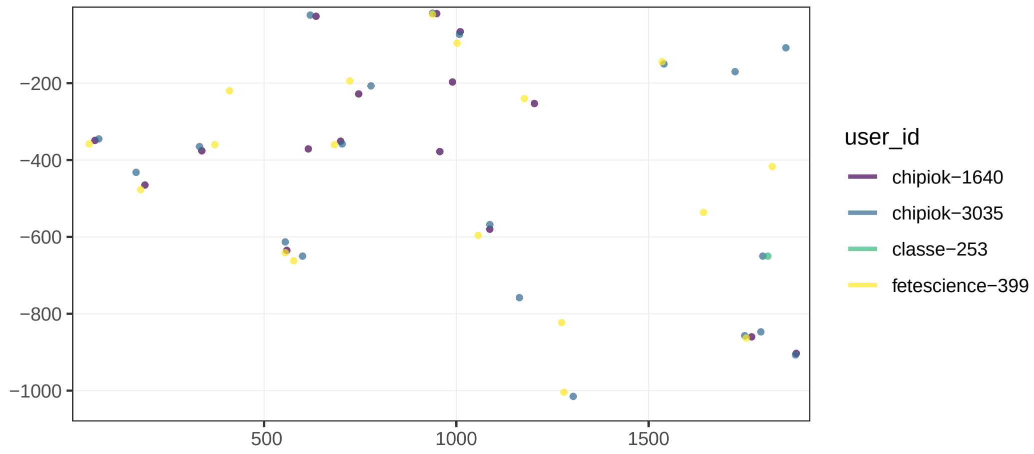
```
#> # A tibble: 268 x 2
#>    image_id     n
#>       <int> <int>
#>  1    12738    46
#>  2    13749    46
#>  3    13059    45
#>  4    12686    42
#>  5    13836    40
#>  6    14075    40
#>  7    12954    39
#>  8    11962    38
#>  9    12024    36
#> 10    13807    34
#> # ... with 258 more rows
```



4.6.4 Multiple users annotations

- Example with `image_id = 11425`

```
gg_users_image(x = ONC2_pyc_carto,
               image_id = 11425)
```

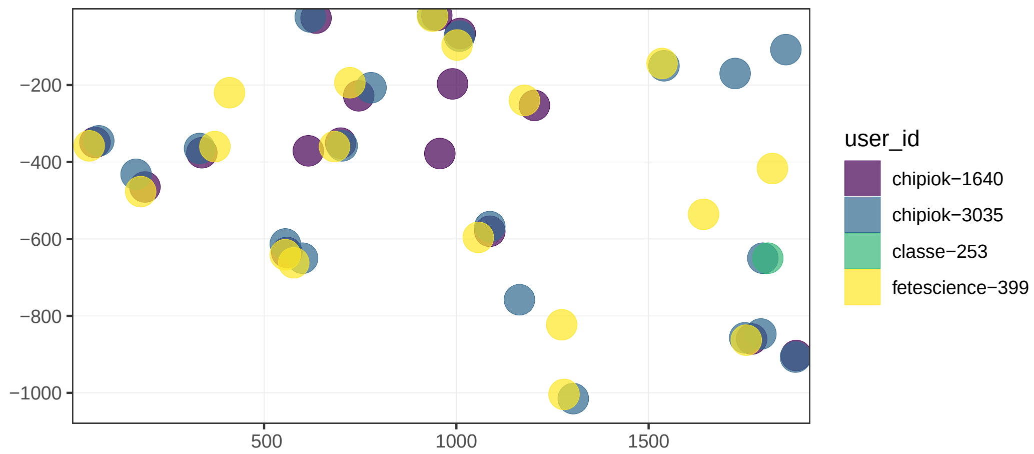


- Comparison with buffer

- Define a buffer size for future analyses

  It seems that precision is low for this point identification, then buffer size need to be big

```
gg_users_image(x = ONC2_pyc_carto,
               image_id = 11425, buffer = 40)
```



4.7. Find all groups

As shown for “Buccinides”, only one function is required to determine the differents groups of annotations in all images. This requires function `find_groups_in_all_images`, some time of computation and RAM available.

```
# Chunk not evaluated in Rmd as results are saved
ONC2_pyc_carto_groups <- find_groups_in_all_images(ONC2_pyc_carto, .progress = TRUE, keep_li

if (!dir.exists(here::here("inst/outputs"))) {
  dir.create("inst/outputs", recursive = TRUE)
}

readr::write_rds(
  ONC2_pyc_carto_groups,
  here::here("inst/outputs", "ONC2_pyc_carto_groups.rds"),
  compress = "gz")
```

```
outwd <- system.file("outputs", package = "deeptools")
ONC2_pyc_carto_groups <- readr::read_rds(file.path(outwd, "ONC2_pyc_carto_groups.rds"))
```

4.8. Calculate statistics on images

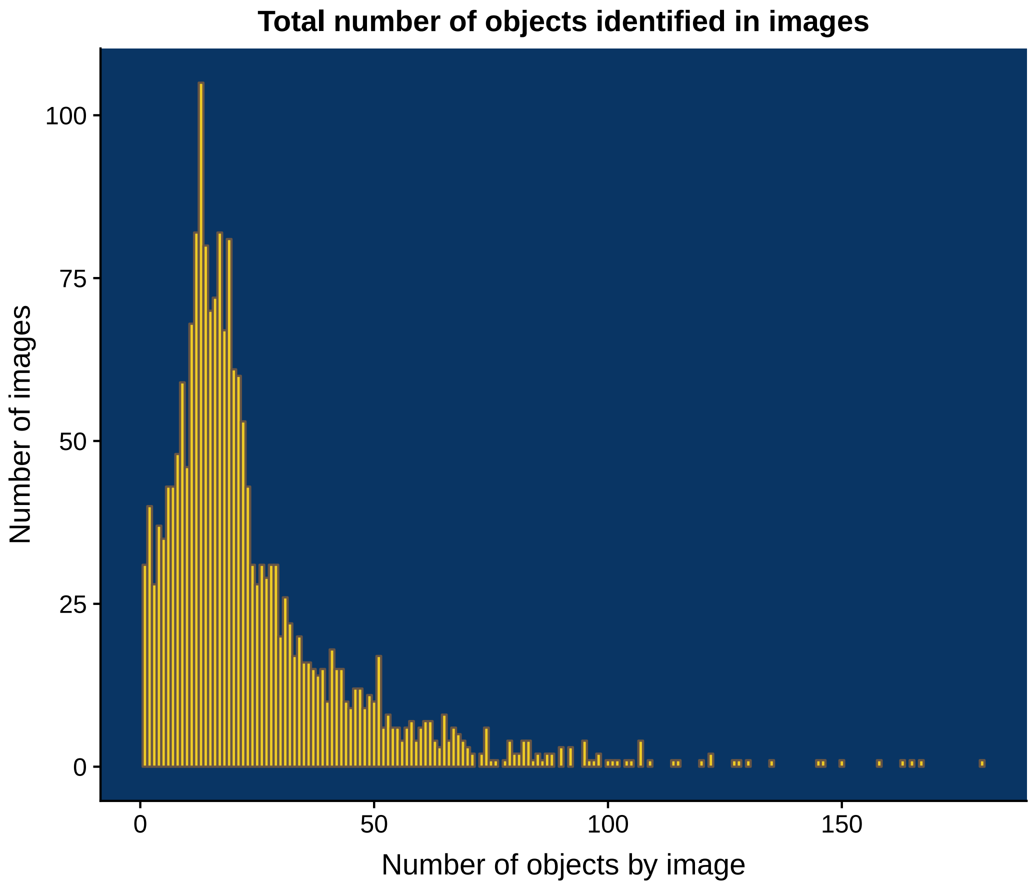
As a reminder, a group of objects is supposed to be a unique individual.

4.8.1 Number of objects per image (already known before)

```
# Number of objects per image (already known before)
pyc_nobjects <- ONC2_pyc_carto_groups %>%
  count(image_id) %>%
  rename(n_objects = n) %>%
  count(n_objects) %>%
  arrange(desc(n)) %>%
  rename(n_images = n)

# Number of marked objects by images
pyc_nobjects
```

```
#> # A tibble: 115 x 2
#>    n_objects n_images
#>        <int>    <int>
#>  1        13      105
#>  2        12       82
#>  3        17       82
#>  4        19       81
#>  5        14       80
#>  6        16       72
#>  7        15       70
#>  8        11       68
#>  9        18       67
#> 10        20       61
#> # ... with 105 more rows
```

```
# Plot
ggplot(pyc_nobjects) +
  geom_col(aes(x = n_objects, y = n_images), width = 1,
           fill = yellow, colour = grey) +
  ggtitle("Total number of objects identified in images") +
  xlab("Number of objects by image") +
  ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



4.8.2 Statistics on groups

Calculate the number of users who marked the same individual. The `proportion` column can be used as a threshold to decide if the individual is kept. Indeed, an individual observed by only one of the users may not be a real individual.

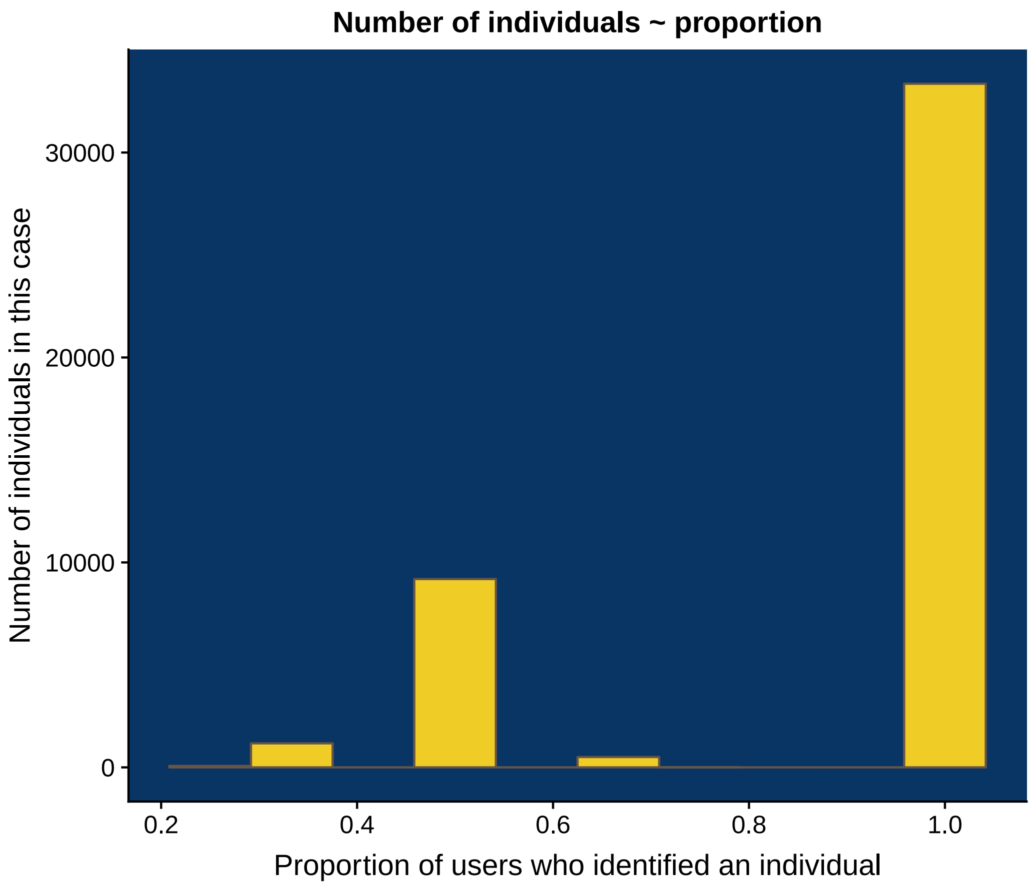
```
# Stats on groups
pyc_groups <- ONC2_pyc_carto_groups %>%
  group_by(image_id, group_kept) %>%
  summarise(
    n_users = n(),
    n_user_id = mean(n_user_id),
    proportion = n()/mean(n_user_id)
  ) %>%
  ungroup()

# Number of individuals ~ proportion
pyc_groups
```

```
#> # A tibble: 44,300 x 5
#>    image_id group_kept n_users n_user_id proportion
#>       <int> <chr>        <int>     <dbl>      <dbl>
#>  1    10680 1                1         1          1
#>  2    10680 10               1         1          1
#>  3    10680 11               1         1          1
#>  4    10680 12               1         1          1
#>  5    10680 13               1         1          1
#>  6    10680 14               1         1          1
#>  7    10680 15               1         1          1
#>  8    10680 16               1         1          1
#>  9    10680 17               1         1          1
```

```
#> 10    10680  18                1          1           1
#> # ... with 44,290 more rows
```

```
# Plot
pyc_groups %>%
  ggplot() +
  geom_histogram(aes(proportion), bins = 10,
                 fill = yellow, colour = grey) +
  ggtitle("Number of individuals ~ proportion") +
  xlab("Proportion of users who identified an individual") +
  ylab("Number of individuals in this case") +
  theme(panel.background = element_rect(fill = blue))
```

```
#> Warning: Removed 1 rows containing non-finite values (stat_bin).
```



4.8.3 Statistics on number of groups by image

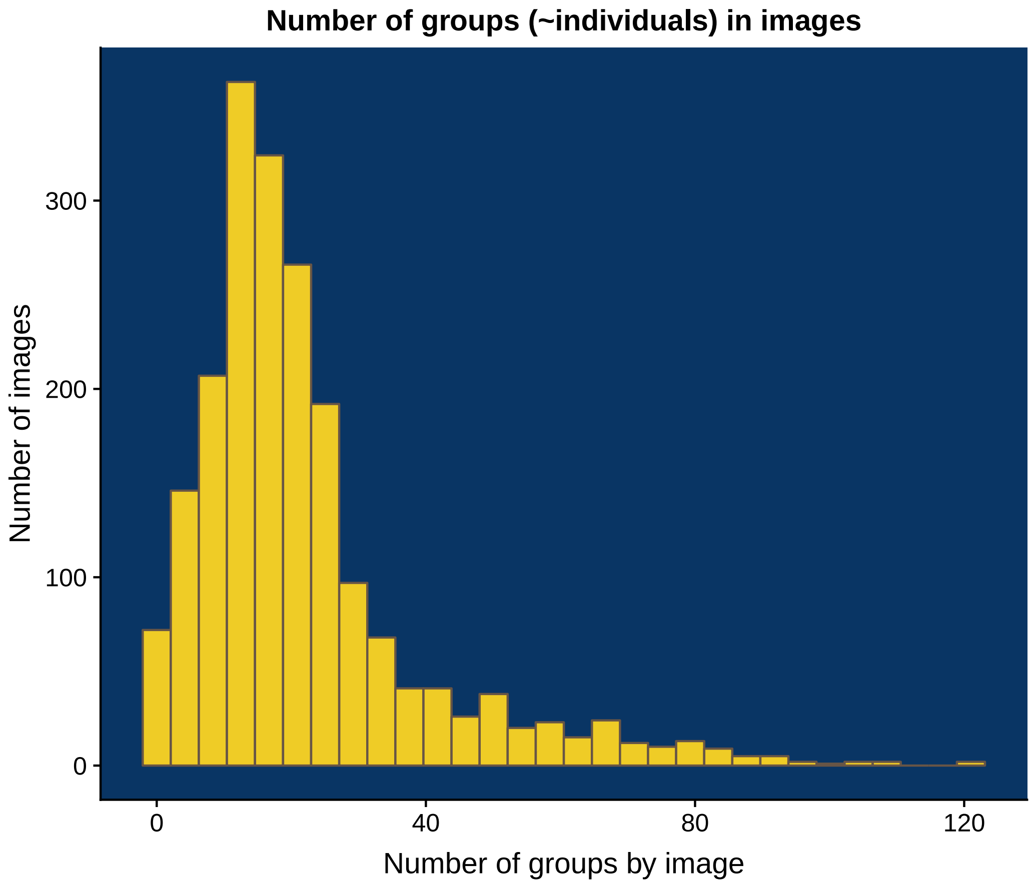
Calculate the number of groups in all images.

```
## Stats on nb groups by image
pyc_ngroups_count <- pyc_groups %>%
  group_by(image_id) %>%
  summarise(n_groups = n())

# Number of images
pyc_ngroups_count %>% count(n_groups)
```

```
#> # A tibble: 100 x 2
#>    n_groups     n
#>       <int> <int>
#>  1        1    32
#>  2        2    40
```

```
#>   3         3    28
#>   4         4    37
#>   5         5    35
#>   6         6    46
#>   7         7    45
#>   8         8    48
#>   9         9    60
#>  10        10    54
#> # ... with 90 more rows
```

```
# Plot
ggplot(pyc_ngroups_count) +
  geom_histogram(aes(x = n_groups), bins = 30,
                 fill = yellow, colour = grey) +
  ggtitle("Number of groups (~individuals) in images") +
  xlab("Number of groups by image") + ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



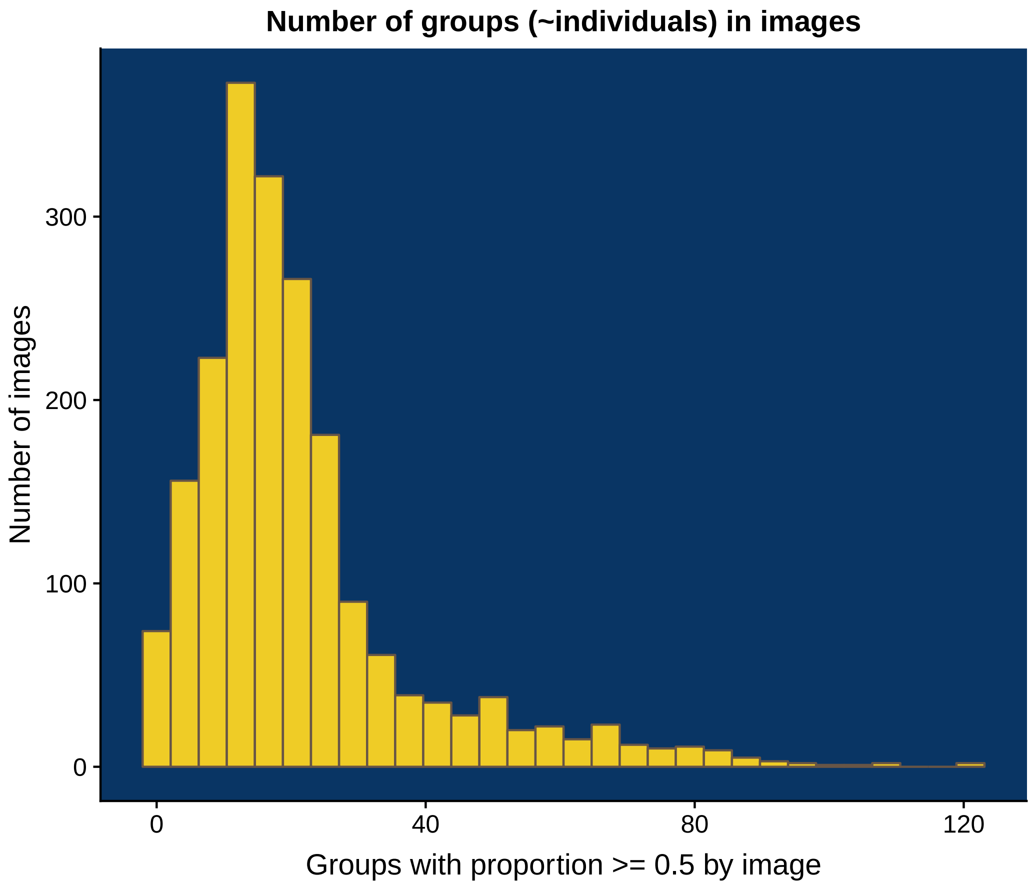
If we only keep groups identified by at least half of the users, we can recalculate the number of groups by image.

```
## Stats on nb groups by image
pyc_ngroups_count_thd <- pyc_groups %>%
  filter(proportion >= 0.5) %>%
  group_by(image_id) %>%
  summarise(n_groups = n())

# Number of images
pyc_ngroups_count_thd %>% count(n_groups)
```

```
#> # A tibble: 98 x 2
#>    n_groups     n
#>       <int> <int>
```

```
#>  1        1    33
#>  2        2    41
#>  3        3    29
#>  4        4    41
#>  5        5    37
#>  6        6    49
#>  7        7    48
#>  8        8    53
#>  9        9    64
#> 10       10    58
#> # ... with 88 more rows
```

```
# Plot
ggplot(pyc_ngroups_count_thd) +
  geom_histogram(aes(x = n_groups), bins = 30,
                 fill = yellow, colour = grey) +
  ggtitle("Number of groups (~individuals) in images") +
  xlab("Groups with proportion >= 0.5 by image") + ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



5. Exploration of "Couverture de moules" data

5.1. Packages

```
library(dplyr)
library(ggplot2)
# devtools::install_github("r-spatial/sf")
library(sf)
library(deeptools)
library(lubridate)
```

```
library(thinkr)
library(readr)
library(tidyr)
```

5.2. Colours

```
blue <- "#093564"
yellow <- "#efcc26"
grey <- "#675546"
```

5.3. Load data

```
# load data
export_file <- system.file("data_orig/export_last.csv", package = "deeptools")
liste_photo <- system.file("data_orig/liste_photo.txt", package = "deeptools")
```

5.4. Prepare data

- Cleaning of species names to be easily usable
- Add user_id combining username and date of image analysis just in case a user sees the same image two times.

```
mission2 <- read_csv(export_file) %>%
  dplyr::select(-comment) %>%
  tidyr::extract(name,
          into = "datetime", regex = "_([[:digit:]]+).",
          remove = FALSE
  ) %>%
  mutate(datetime = ymd_hms(datetime, tz = "UTC")) %>%
  # clean names of species
  mutate(name_fr_clean = clean_vec(name_fr, unique = FALSE)) %>%
  group_by(username) %>%
  mutate(
    user_id = paste(username, as.character(as.numeric(as.factor(datDeb))), sep = "-")
  ) %>%
  ungroup()
```

```
#> Parsed with column specification:
#> cols(
#>   id = col_integer(),
#>   image_id = col_integer(),
#>   name = col_character(),
#>   username = col_character(),
#>   userlevel = col_integer(),
#>   comment = col_character(),
#>   datDeb = col_datetime(format = ""),
#>   datFin = col_datetime(format = ""),
#>   obs_code = col_character(),
#>   name_fr = col_character(),
#>   pos1x = col_integer(),
#>   pos1y = col_integer(),
#>   pos2x = col_integer(),
#>   pos2y = col_integer(),
#>   length = col_integer(),
#>   middle_x = col_integer(),
#>   middle_y = col_integer(),
```

```
#>   polygon_values = col_character()
#> )

#> Warning in rbind(names(probs), probs_f): number of columns of result is not
#> a multiple of vector length (arg 1)

#> Warning: 305088 parsing failures.
#> row # A tibble: 5 x 5 col     row col     expected  actual file
#> ... ................. ... ...............................................................
#> See problems(...) for more details.
```

```
# Separate observatory dataset
mission2_MAR <- mission2 %>% filter(obs_code == "MAR")
mission2_ONC <- mission2 %>% filter(obs_code == "JDF")
```

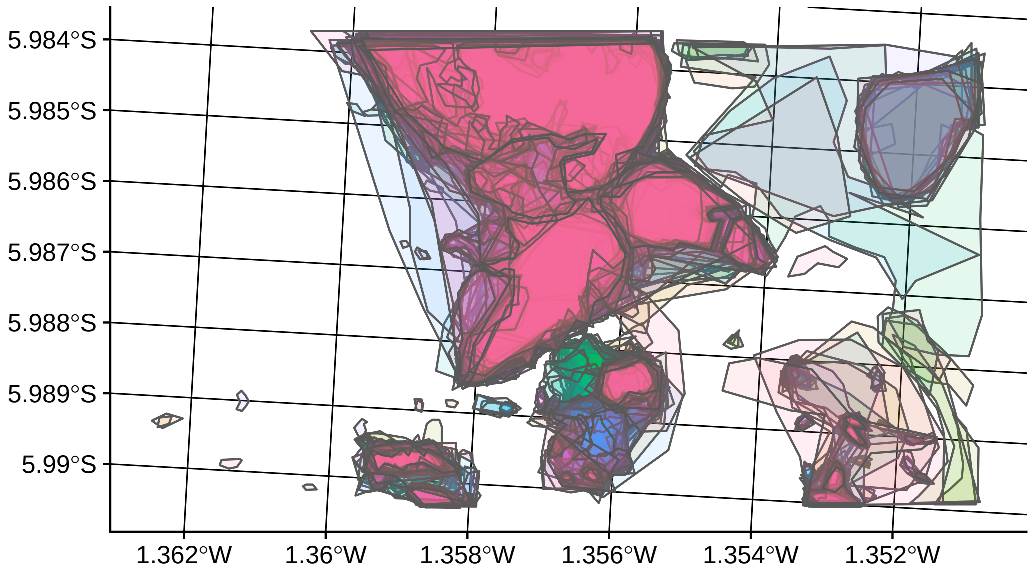
5.5. Extraction of "Buccinide"

Function `to_carto` extract and transform data as spatial object for following analyses.

```
# Filter on Buccinide only
MAR_mussel <- mission2_MAR %>%
  filter(name_fr_clean == "couverture_de_moules")

# Filter and transform as spatial data
MAR_mussel_carto <- mission2_MAR %>%
  filter(name_fr_clean == "couverture_de_moules") %>%
  to_carto(name_fr_clean, "couverture_de_moules")
```

Only because it is nice

```
ggplot(MAR_mussel_carto) +
  geom_sf(aes(fill = as.character(image_id)), alpha = 0.1) +
  guides(fill = FALSE)
```



5.6. Exploration of annotations

5.6.1 Users

```
MAR_mussel %>%
  count(username) %>%
  arrange(desc(n))
```

```
#> # A tibble: 2 x 2
#>   username      n
#>   <chr>     <int>
#> 1 chipiok    1527
#> 2 grillus33    71
```

5.6.2 Images

- Number of annotations by image

```
MAR_mussel %>%
  count(image_id) %>%
  arrange(desc(n))
```

```
#> # A tibble: 133 x 2
#>    image_id     n
#>       <int> <int>
#>  1    14411    32
#>  2    14589    28
#>  3    14186    27
#>  4    14307    25
#>  5    14281    23
#>  6    14326    23
#>  7    14396    21
#>  8    14552    21
#>  9    14352    20
#> 10    14147    19
#> # ... with 123 more rows
```

- Number of users by image

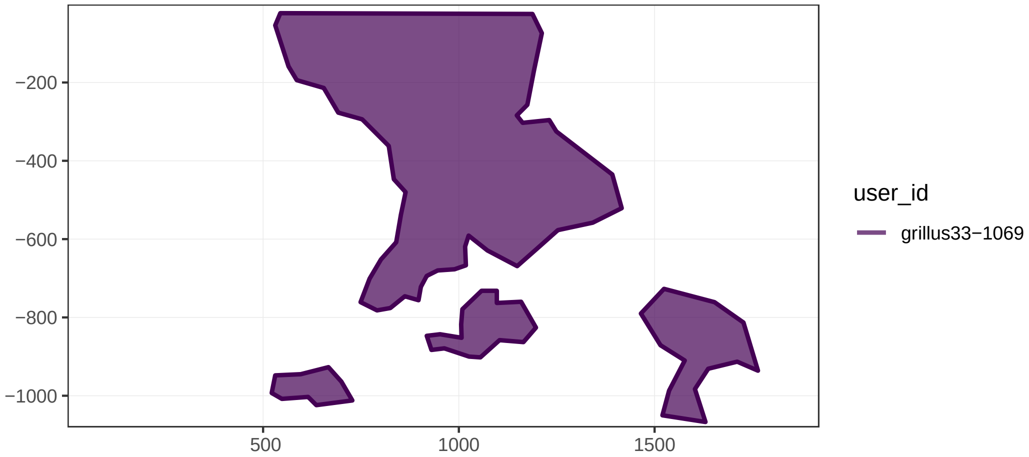
```
MAR_mussel %>%
  group_by(image_id) %>%
  summarize(n_users = length(unique(user_id))) %>%
  arrange(desc(n_users))
```

```
#> # A tibble: 133 x 2
#>    image_id n_users
#>       <int>   <int>
#>  1    14186       2
#>  2    14190       2
#>  3    14281       2
#>  4    14307       2
#>  5    14326       2
#>  6    14329       2
#>  7    14352       2
#>  8    14387       2
#>  9    14396       2
#> 10    14411       2
#> # ... with 123 more rows
```

5.6.3 Example: `filter_col = username, filter_val = "grillus33", image_id == "14190"`

```
#> # A tibble: 18 x 2
#>    image_id     n
#>       <int> <int>
#>  1    14191     5
#>  2    14608     5
#>  3    14623     5
```

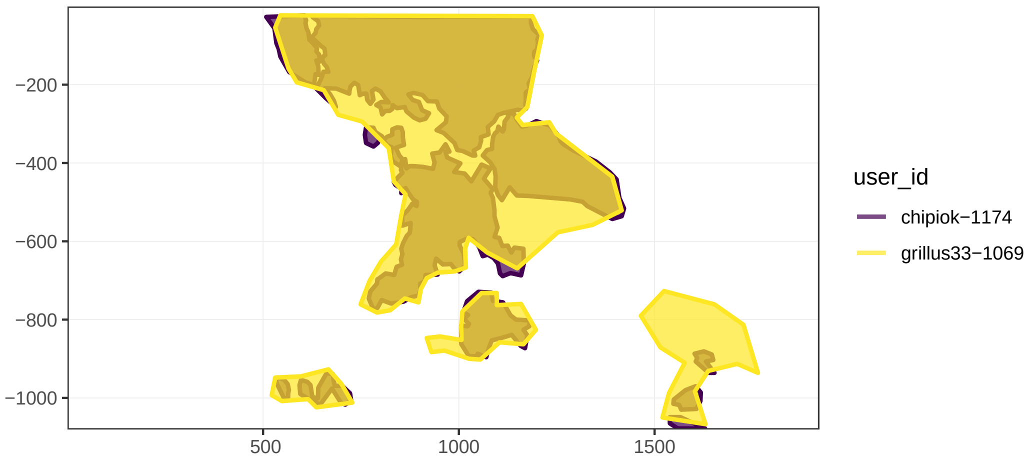
```
#>   4     14635      5
#>   5     14654      5
#>   6     14184      4
#>   7     14190      4
#>   8     14286      4
#>   9     14358      4
#>  10     14387      4
#>  11     14470      4
#>  12     14535      4
#>  13     14132      3
#>  14     14281      3
#>  15     14329      3
#>  16     14396      3
#>  17     14469      3
#>  18     14636      3
```



5.6.4 Multiple users annotations

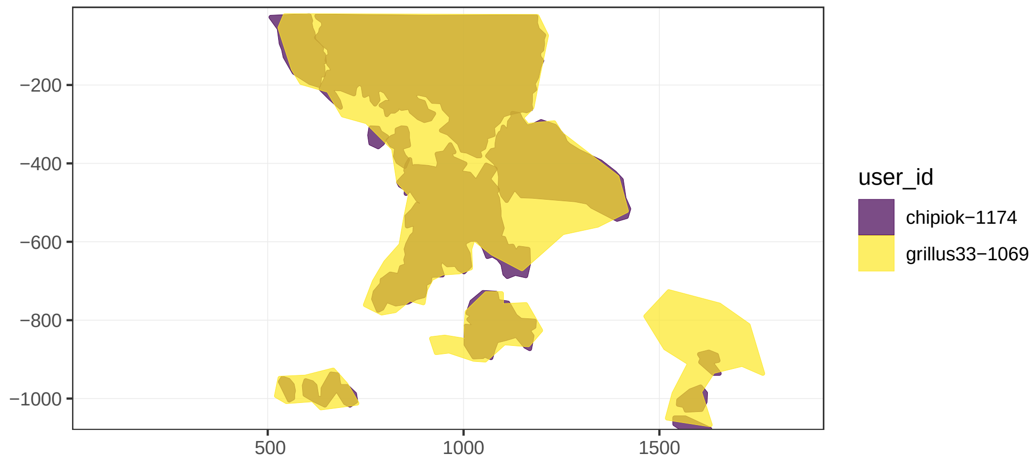
- Example with `image_id = 14190`

```
gg_users_image(x = MAR_mussel_carto,
               image_id = 14190)
```



- Because we work with polygons, buffer is not totally necessary. However, just in case there are small areas, we can set a small buffer.

```
gg_users_image(x = MAR_mussel_carto,
               image_id = 14190, buffer = 5)
```



5.7. Find all groups

As shown for "Buccinides", only one function is required to determine the differents groups of annotations in all images. This requires function `find_groups_in_all_images`, some time of computation and RAM available.

```
# Chunk not evaluated in Rmd as results are saved
MAR_mussel_carto_groups <- MAR_mussel_carto %>%
  find_groups_in_all_images(.progress = TRUE, keep_list = FALSE,
                            as_sf = FALSE, dist_buffer = 5)

if (!dir.exists(here::here("inst/outputs"))) {
  dir.create("inst/outputs", recursive = TRUE)
}

readr::write_rds(
  MAR_mussel_carto_groups,
  here::here("inst/outputs", "MAR_mussel_carto_groups.rds"),
  compress = "gz")
```

```
outwd <- system.file("outputs", package = "deeptools")
MAR_mussel_carto_groups <- readr::read_rds(file.path(outwd, "MAR_mussel_carto_groups.rds"))
```

5.8. Calculate statistics on images

As a reminder, a group of objects is supposed to be a unique individual.

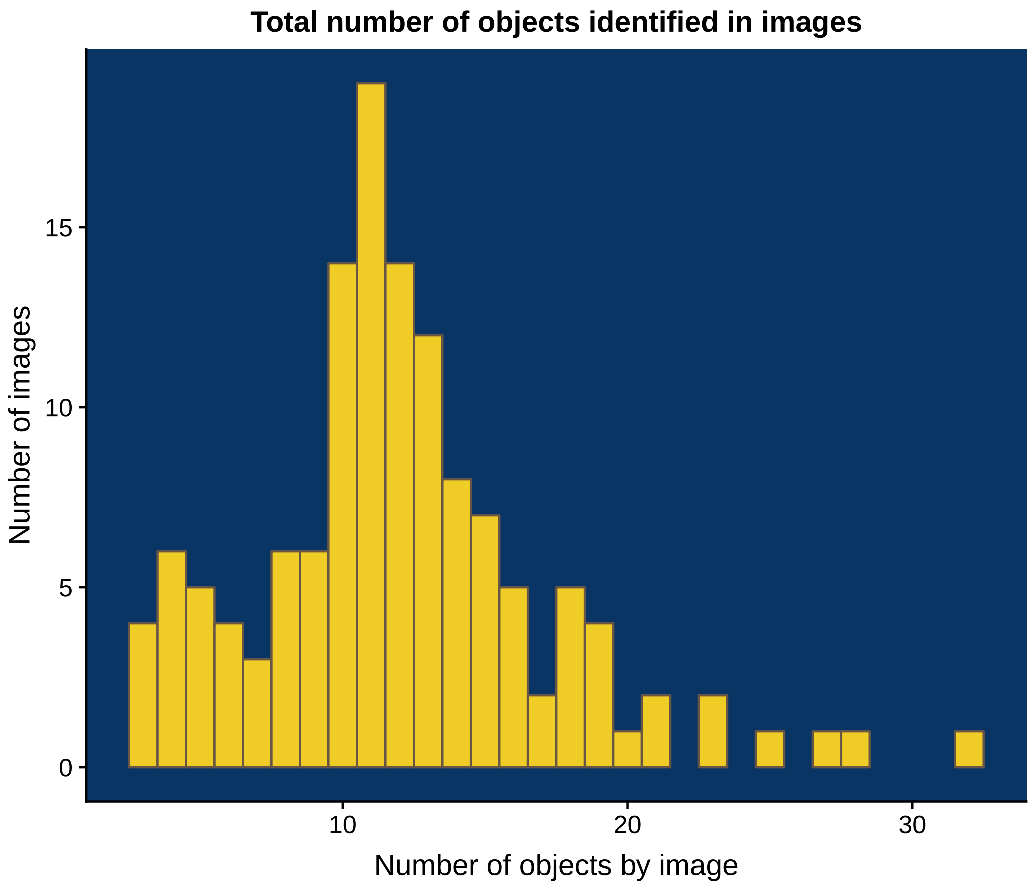
5.8.1 Number of objects per image (already known before)

```
# Number of objects per image (already known before)
mussel_nobjects <- MAR_mussel_carto_groups %>%
  count(image_id) %>%
  rename(n_objects = n) %>%
  count(n_objects) %>%
  arrange(desc(n)) %>%
  rename(n_images = n)

# Number of marked objects by images
mussel_nobjects
```

```
#> # A tibble: 24 x 2
#>    n_objects n_images
```

```
#>          <int>    <int>
#>   1         11       19
#>   2         10       14
#>   3         12       14
#>   4         13       12
#>   5         14        8
#>   6         15        7
#>   7          4        6
#>   8          8        6
#>   9          9        6
#>  10          5        5
#> # ... with 14 more rows
```

```
# Plot
ggplot(mussel_nobjects) +
  geom_col(aes(x = n_objects, y = n_images), width = 1,
           fill = yellow, colour = grey) +
  ggtitle("Total number of objects identified in images") +
  xlab("Number of objects by image") +
  ylab("Number of images") +
  theme(panel.background = element_rect(fill = blue))
```



5.8.2 Calculate surfaces of polygons

- We calculte the average and the standard deviation of groups

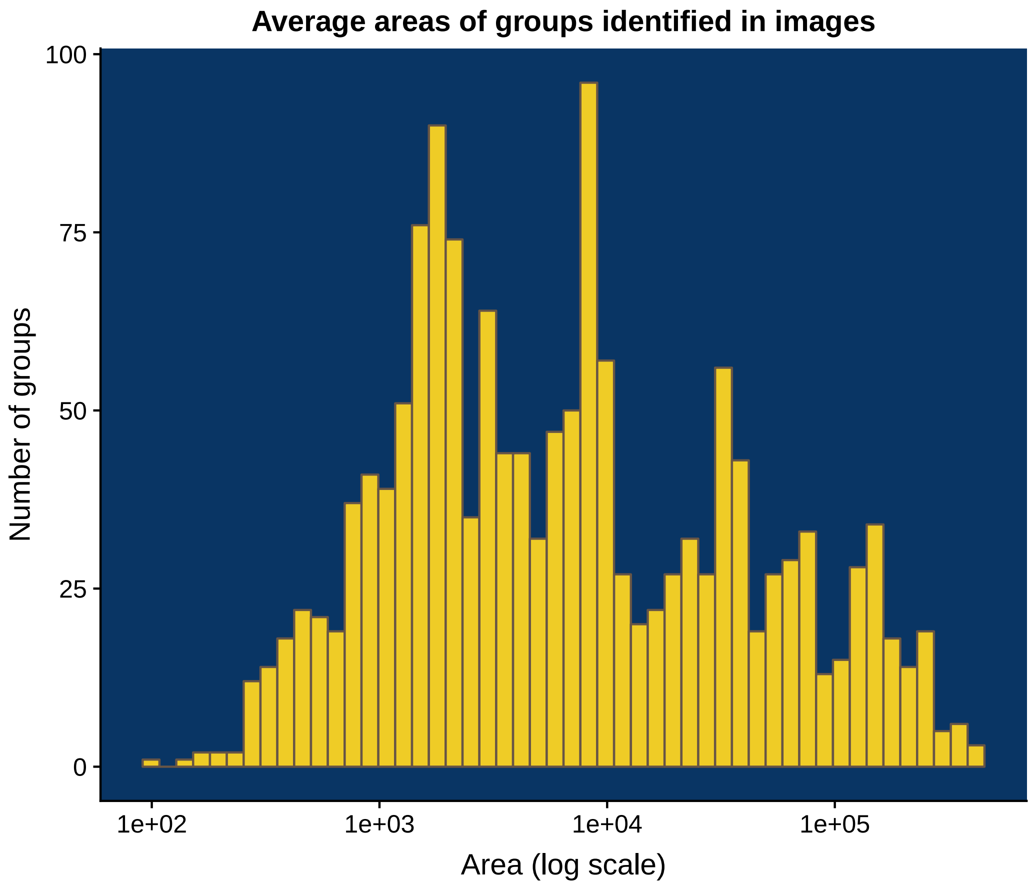
```
MAR_mussel_carto_groups_area <- MAR_mussel_carto_groups %>%
  mutate(area = MAR_mussel_carto_groups %>%
           st_sf() %>% st_area()) %>%
  group_by(image_id, group_kept) %>%
  summarise(nb_polygons_in_group = n(),
            area_mean = mean(area),
```

```
             area_sd = sd(area),
             area_sd = if_else(is.na(area_sd), 0, area_sd),
             area_cv = area_sd / area_mean) %>%
  arrange(desc(nb_polygons_in_group))

MAR_mussel_carto_groups_area
```

```
#> # A tibble: 1,508 x 6
#> # Groups:   image_id [133]
#>    image_id group_kept nb_polygons_in_group area_mean  area_sd area_cv
#>       <int> <chr>                     <int>     <dbl>    <dbl>   <dbl>
#>  1    14186 1-16                          2     1656.    211.    0.127
#>  2    14186 11-20                         2   176066.  41803.    0.237
#>  3    14186 12-22                         2    31141.   4061.    0.130
#>  4    14186 13-23                         2     5800.   4407.    0.760
#>  5    14186 2-17                          2      285.    159.    0.557
#>  6    14186 3-15                          2     2379.     89.4   0.0376
#>  7    14186 4-18                          2     1452     135.    0.0930
#>  8    14186 5-26                          2      823.      9.55  0.0116
#>  9    14186 6-27                          2      608.    319.    0.525
#> 10    14186 7-14                          2     8050     142.    0.0177
#> # ... with 1,498 more rows
```

- Graph of areas

```
MAR_mussel_carto_groups_area %>%
  ggplot() +
  geom_histogram(aes(area_mean), bins = 50,
           fill = yellow, colour = grey) +
  ggtitle("Average areas of groups identified in images") +
  xlab("Area (log scale)") +
  ylab("Number of groups") +
  theme(panel.background = element_rect(fill = blue)) +
  scale_x_log10()
```



6. TODO

- [x] to_carto_point() : ONC => "pycnogonide"
- [x] to_carto_polygon() : MAR => "couverture_de_moules"
- [x] Flip-y en paramètre
- [] Séparer l'analyse des images zoomées ou non zoomées.
    - Influence sur le choix du buffer de comparaison
- [x] Choose statistics based on occurence
    - threshold: If a group is found by only 1/5 users, do we remove it ? Do we have to define this threshold according to userlevel ?

7. List of dependencies

- bookdown (Xie (2018a))
- cowplot (Wilke (2018))
- dplyr (Wickham et al. (2018b))
- fasterize (Ross (2018))
- forcats (Wickham (2018a))
- furrr (Vaughan and Dancho (2018))
- future (Bengtsson (2018))
- ggplot2 (Wickham et al. (2018a))
- ggrepel (Slowikowski (2018))
- igraph (file. (2018))
- knitr (Xie (2018b))
- lubridate (Spinu et al. (2018))
- magrittr (Bache and Wickham (2014))
- pkgdown (Wickham and Hesselberth (2018))
- purrr (Henry and Wickham (2018))

- raster (Hijmans (2018))
- rasterVis (Perpinan Lamigueiro and Hijmans (2018))
- readr (Wickham et al. (2017))
- rlang (Henry and Wickham (2019))
- rmarkdown (Allaire et al. (2018))
- sf (Pebesma (2018))
- stats (**?**)
- stringr (Wickham (2018b))
- thinkr (Guyader and Rochette (2018))
- tidyr (Wickham and Henry (2018))
- utils (**?**)

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.11.

Bache, S. M. and Wickham, H. (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5.

Bengtsson, H. (2018). *future: Unified Parallel and Distributed Processing in R for Everyone*. R package version 1.10.0.

file., S. A. (2018). *igraph: Network Analysis and Visualization*. R package version 1.2.2.

Guyader, V. and Rochette, S. (2018). *thinkr: Tools for Cleaning Up Messy Files*. R package version 0.12.

Henry, L. and Wickham, H. (2018). *purrr: Functional Programming Tools*. R package version 0.2.5.

Henry, L. and Wickham, H. (2019). *rlang: Functions for Base Types and Core R and 'Tidyverse' Features*. R package version 0.3.1.

Hijmans, R. J. (2018). *raster: Geographic Data Analysis and Modeling*. R package version 2.8-4.

Pebesma, E. (2018). *sf: Simple Features for R*. R package version 0.7-1.

Perpinan Lamigueiro, O. and Hijmans, R. (2018). *rasterVis: Visualization Methods for Raster Data*. R package version 0.45.

Ross, N. (2018). *fasterize: Fast Polygon to Raster Conversion*. R package version 1.0.0.

Slowikowski, K. (2018). *ggrepel: Automatically Position Non-Overlapping Text Labels with 'ggplot2'*. R package version 0.8.0.

Spinu, V., Grolemund, G., and Wickham, H. (2018). *lubridate: Make Dealing with Dates a Little Easier*. R package version 1.7.4.

Vaughan, D. and Dancho, M. (2018). *furrr: Apply Mapping Functions in Parallel using Futures*. R package version 0.1.0.

Wickham, H. (2018a). *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 0.3.0.

Wickham, H. (2018b). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.3.1.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., and Woo, K. (2018a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0.

Wickham, H., François, R., Henry, L., and Müller, K. (2018b). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.8.

Wickham, H. and Henry, L. (2018). *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions.* R package version 0.8.2.

Wickham, H. and Hesselberth, J. (2018). *pkgdown: Make Static HTML Documentation for a Package.* R package version 1.3.0.

Wickham, H., Hester, J., and Francois, R. (2017). *readr: Read Rectangular Text Data.* R package version 1.1.1.

Wilke, C. O. (2018). *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'.* R package version 0.9.3.

Xie, Y. (2018a). *bookdown: Authoring Books and Technical Documents with R Markdown.* R package version 0.9.

Xie, Y. (2018b). *knitr: A General-Purpose Package for Dynamic Report Generation in R.* R package version 1.21.