Demo for the presentation 'Getting insights from automatic feeder data'

Luis Gonzalez-Gracia (lagog6@ulaval.ca)   Twitter - @gonzaluisandres

28 September 2022

Abstract

This short R Notebook will show the workflow as presented in my presentation today. We will grab to different sources of data, and do some wrangling, visualization, and analysis. The script can also be accessed in the `demo.R` file, also in this repository.

How to get and run this files on your own computer:

Figure 1: Some of you might be too young to get the reference

You can run and tweak this code by cloning my repository. If you do not know what a repository is, I recommend you to begin reading about it if you want to work collaboratively with other people.

1. Install git https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
2. Go to the folder you want the folder with the files to be located.
3. Open that folder in the terminal and type

```
git clone https://git.disroot.org/luangonz/feeder-analytics-demo.git
```

```
## Cloning into 'feeder-analytics-demo'...
```

Git will automatically create a folder and download all the files required to run this code.

Setup

Since the idea of this project is to be as modular and automated as possible, I created a set of custom functions that do the heavy lifting in the background, without cluttering too much the script file. This has some advantages and disadvantages, since although it is easier to read and the workflow is easy to follow along, if issues appear, then debugging and following the function that originated the issue is more time-consuming.

The folder is organized in the main project files and two folders:

- The `data` folder holds the data files we will be loading into the environment.
- The `setup` folder (used in this section) holds two key files:
  - `functions.R` that holds all the custom functions
  - `loadlibraries.R` that has all the libraries needed for any script file that uses the same set of libraries (so you do not need to copy and paste it in every file of the project)

When we run the `source()` function at this step, we are loading both the libraries and the functions that we are going to use throughout the demo.

```r
source("setup/functions.R")
source("setup/loadlibraries.R")
```

Importing data

First step in the process is to load the excel file in the environment. There are packages that can natively import excel files that are very straightforward, but some of them do not handle the Date information properly. For this I have a custom function that takes into account some common issues and with the `method =` parameter, I can fine-tune the importing method according to where the data comes from.

```r
farmA <- xlsx_to_dataframe(filename = "data/farm_A_demo.xlsx", # selects the file
                           method = "farm_A_11rows") # selects the farm A method
```

Lucky for us, farm B has its data directly formatted in the RData format, which helps a lot in the importing process. A simple `load()` function and the data is there.

```r
load("data/farmB.RData") # load the file into the environment

farmB <- data_alim # renaming to make it a better understandable filename
rm(data_alim) # removing the original imported dataframe
```

Checking structure of data

We will look at the raw imported data as it comes from the import procedure.

```
str(farmA)
```

```
## tibble [268,199 x 10] (S3: tbl_df/tbl/data.frame)
##  $ Date    : POSIXct[1:268199], format: "2021-04-27" "2021-04-27" ...
##  $ Tatouage: chr [1:268199] NA NA NA NA ...
##  $ Station : num [1:268199] 1 1 1 1 1 1 1 1 1 1 ...
##  $ pdsdeb  : num [1:268199] 1.408 1.392 0.778 1.248 1.247 ...
##  $ pdsfin  : num [1:268199] 1.392 1.385 0.771 1.189 1.245 ...
##  $ cons    : num [1:268199] 0.016 0.007 0.007 0.059 0.002 0.02 0.04 0.468 0.261 0.364 ...
##  $ remp    : num [1:268199] 0 0 0 0 0 0 0 0 0 0.48 ...
##  $ hredeb  : chr [1:268199] "4:43:19" "4:44:09" "8:06:28" "8:38:20" ...
##  $ hrefin  : chr [1:268199] "4:43:43" "4:44:14" "8:06:40" "8:40:52" ...
##  $ duree1  : chr [1:268199] "0:00:24" "0:00:05" "0:00:12" "0:02:32" ...
```

We see some issues that are of concern, for example time of start of visit is not in a proper date/time type, but it is only a character. Lets check Farm B:

```
str(farmB)
```

```
## 'data.frame':    32827 obs. of  12 variables:
##  $ Date_fin           : Date, format: "2020-05-12" "2020-05-12" ...
##  $ Animal             : chr  "4817" "4817" "4815" "4817" ...
##  $ RFID               : num  7406487 7406487 13333982 7406487 13333982 ...
##  $ Parc               : int  6 6 6 6 6 6 6 6 6 6 ...
##  $ Poids_aliment_debut: num  13.4 13.4 13.4 13.4 13.4 ...
##  $ Poids_aliment_fin  : num  13.4 13.4 13.4 13.4 13.4 ...
##  $ Qte_aliment        : num  0 0.01 -0.01 0 0 0 0 0 0 0.01 ...
##  $ Tdebut             : chr  "10:28:15" "10:30:06" "10:32:47" "10:34:17" ...
##  $ Tfin               : chr  "10:28:56" "10:32:47" "10:34:17" "10:34:42" ...
##  $ Duree_insentec     : num  0.41 2.41 1.3 0.25 0.36 0.21 0.05 0.1 0.58 0.02 ...
##  $ Dummy              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Duree_s            : num  41 161 90 25 36 21 5 10 58 2 ...
```

Similar issues with the time and also the titles of the variables between these two are different, making it hard to work with them with just a piece of code. So the strategy is to take this (or any) kind of dataframe that we work with, and standardize it to a format that any of the next functions can work with. Next step then, is standardization.

Standardization

The `harmonize_feeder_data()` function is a custom function that allows us to funnel any kind of source file into a single, homogenous data structure so it can be fed into the following functions in the workflow. It has two parameters:

- `groupstations`: If TRUE, the station number becomes a group in the dataframe (useful for summarizations).

- `method`: a selector for the method that it will use, according to which source the data frame comes from
- `remove_filling`: if TRUE, it will remove the FILLING events of the feeder (when the feeder is filled up).
- `remove_na`: if TRUE, it will remove unavailable data that might interfere in some of the calculations.

```
farmB_standard <- harmonize_feeder_data(farmB,
                                        groupstations = TRUE,
                                        method = "deschambault")

farmA_standard <- harmonize_feeder_data(farmA,
                                             groupstations = TRUE,
                                             method = "farm_A_raw",
                                             remove_filling = TRUE,
                                             remove_na = TRUE)
```

We will check the structure again to see if everything is in order:

```
str(farmA_standard)
```

```
## tibble [261,636 x 9] (S3: tbl_df/tbl/data.frame)
##  $ Date          : POSIXct[1:261636], format: "2021-04-27" "2021-04-27" ...
##  $ id            : Factor w/ 308 levels "SOGE60281J","SOGE60286J",..: 4 4 4 4 7 7 7 7 7 7 ...
##  $ station       : Factor w/ 22 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ pdsdeb        : num [1:261636] 1.282 1.068 0.991 1.03 1.448 ...
##  $ pdsfin        : num [1:261636] 0.814 0.807 0.627 0.524 1.282 ...
##  $ cons          : num [1:261636] 0.468 0.261 0.364 0.506 0.166 0.265 0.209 0.092 0.122 0.295 ...
##  $ hour.in       : POSIXct[1:261636], format: "2021-04-27 04:56:28" "2021-04-27 08:54:00" ...
##  $ hour.out      : POSIXct[1:261636], format: "2021-04-27 05:15:48" "2021-04-27 09:05:44" ...
##  $ visit_dur_secs: 'difftime' num [1:261636] 1160 704 1306 1651 ...
##   ..- attr(*, "units")= chr "secs"
```

```
str(farmB_standard)
```

```
## 'data.frame':    26853 obs. of  9 variables:
##  $ Date          : Date, format: "2020-05-12" "2020-05-12" ...
##  $ id            : Factor w/ 20 levels "4804","4805",..: 14 14 12 14 12 14 1 14 5 5 ...
##  $ station       : Factor w/ 1 level "6": 1 1 1 1 1 1 1 1 1 1 ...
##  $ pdsdeb        : num  13.4 13.4 13.4 13.4 13.4 ...
##  $ pdsfin        : num  13.4 13.4 13.4 13.4 13.4 ...
##  $ cons          : num  0 0.01 -0.01 0 0 0 0 0 0 0.01 ...
##  $ hour.in       : POSIXct, format: "2020-05-12 10:28:15" "2020-05-12 10:30:06" ...
##  $ hour.out      : POSIXct, format: "2020-05-12 10:28:56" "2020-05-12 10:32:47" ...
##  $ visit_dur_secs: 'difftime' num  41 161 90 25 ...
##   ..- attr(*, "units")= chr "secs"
```

With this function we‘ve managed to homogenize the data structure so we can move on now to our next step.

Inspecting data integrity

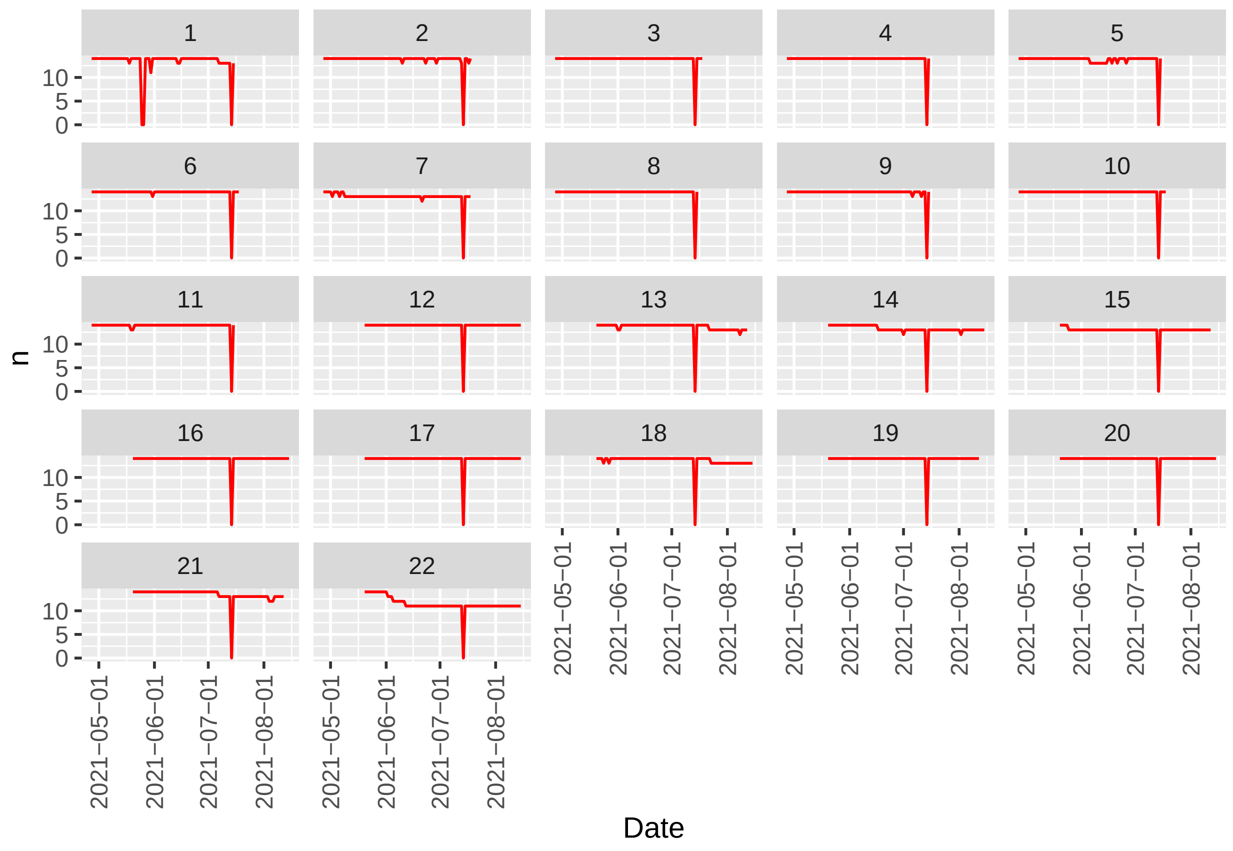
Well be running some more custom functions to plot valuable data.

Population plot

Farm A has 22 different pens. It would be valuable to see if there are any issues regarding the population of these pens, for example a quick reduction or increase in size or a quick drop due to data loss form the hardware

Population plot of farm A

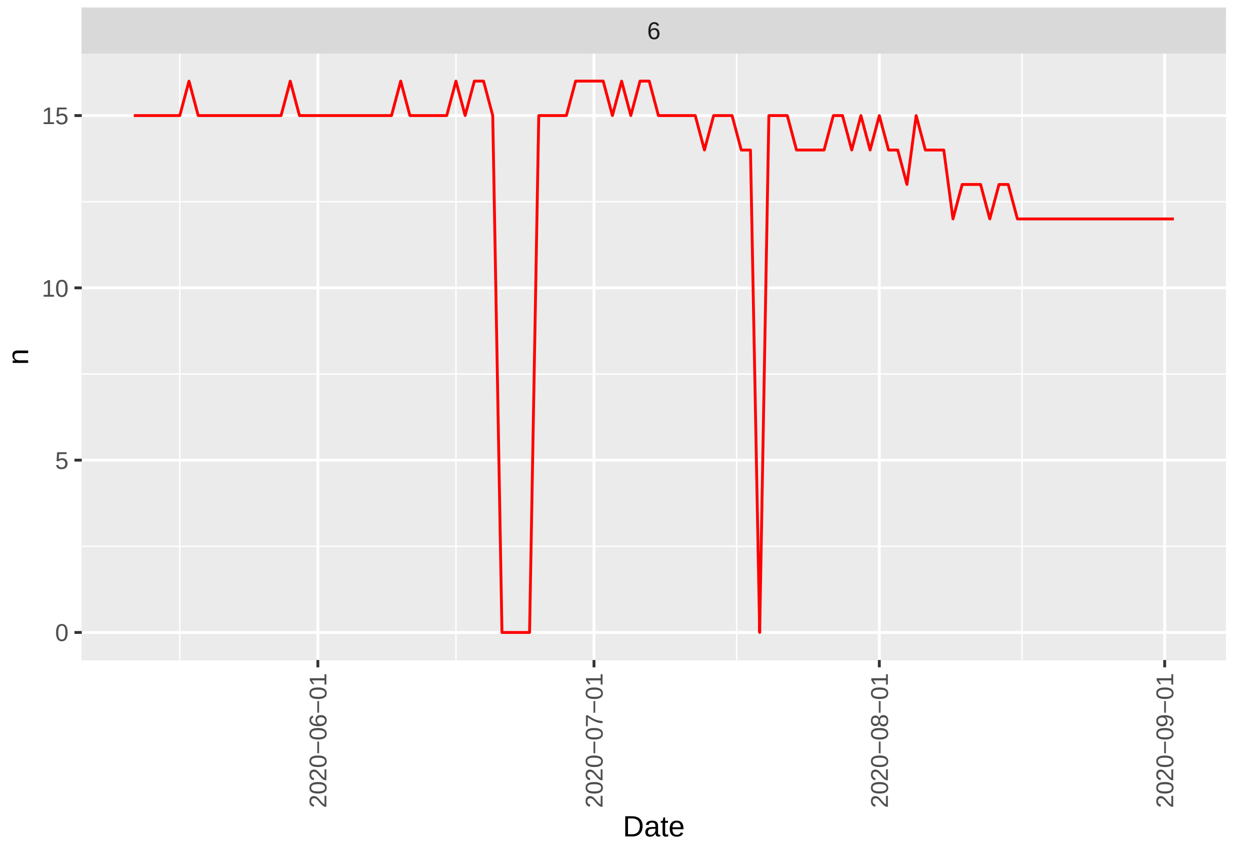
```
populationPlot(farmA_standard)
```



Population plot of farm B

```
populationPlot(farmB_standard)
```

```
## ‘summarise()‘ has grouped output by ’station’. You can override using the
## ‘.groups‘ argument.
## pad applied on the interval: day
```

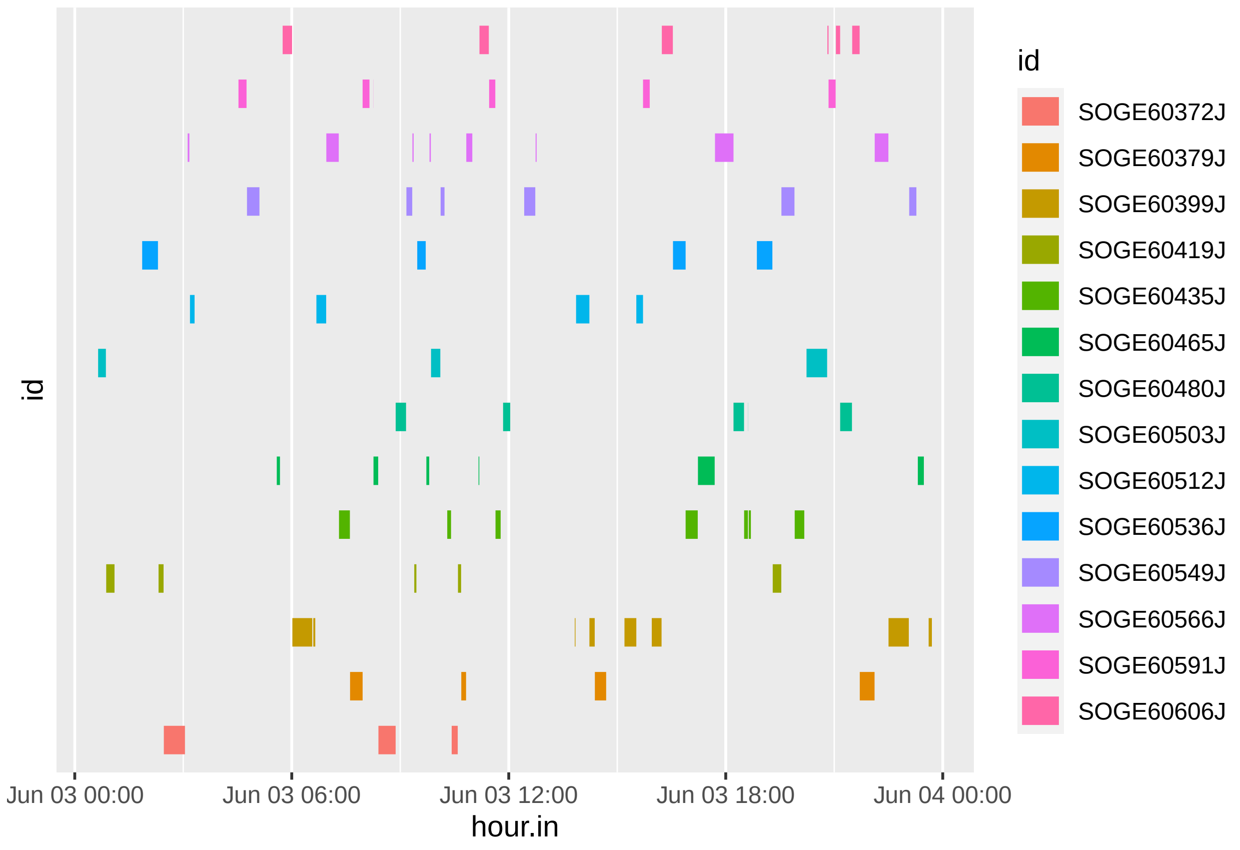


We can evidence with these plots that there are some pen size fluctuations and some data loss in some of the periods. These losses will need to be taken into account during the analyses.

Visualizing visits to the feeder

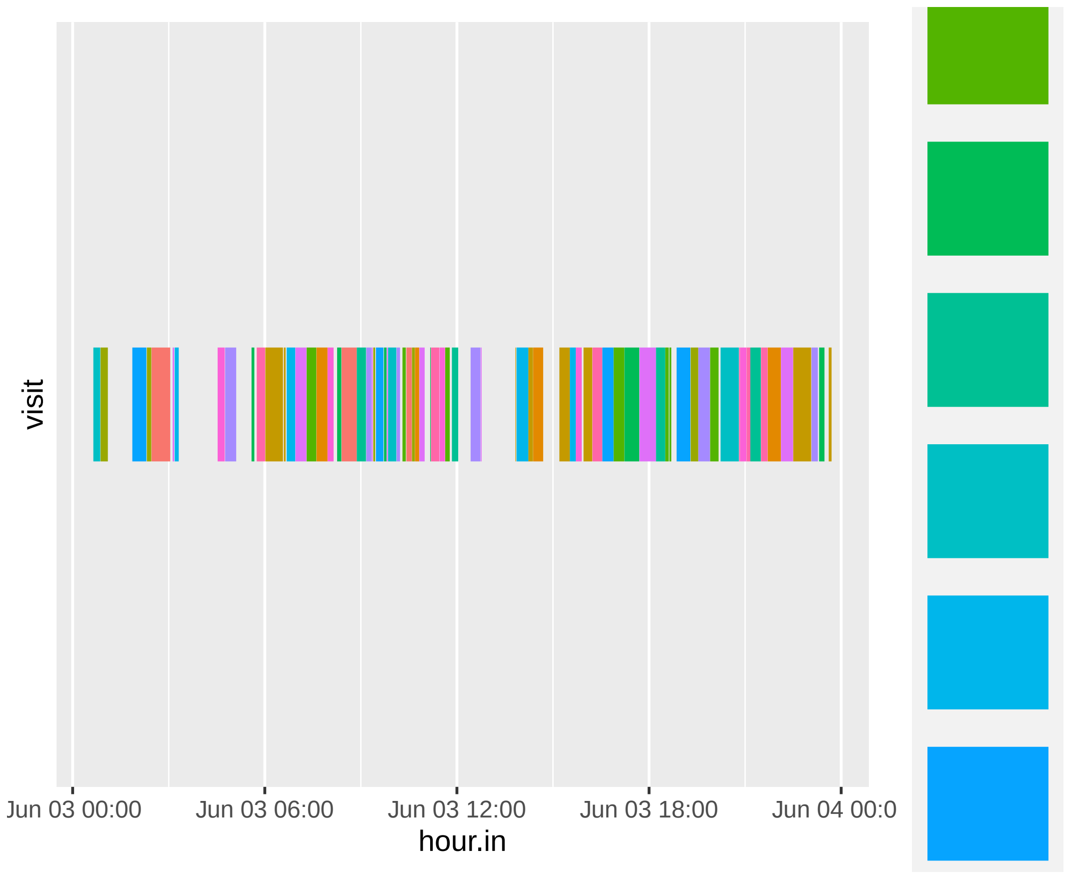
We can visualize a timeline of visits to the feeder for any station or day with this custom function, `visitPlotsDay()`:

```
visitPlotsDay(farmA_standard,
              thedate = "2021-06-03",
              thestation = 11,
              singlestrip = FALSE)
```



With the last plot, we have one line per pig, but sometimes seeing all the visit in a single line is useful. This is what the `singlestrip` parameter is useful for.

```r
visitPlotsDay(farmA_standard,
              thedate = "2021-06-03",
              thestation = 11,
              singlestrip = TRUE)
```

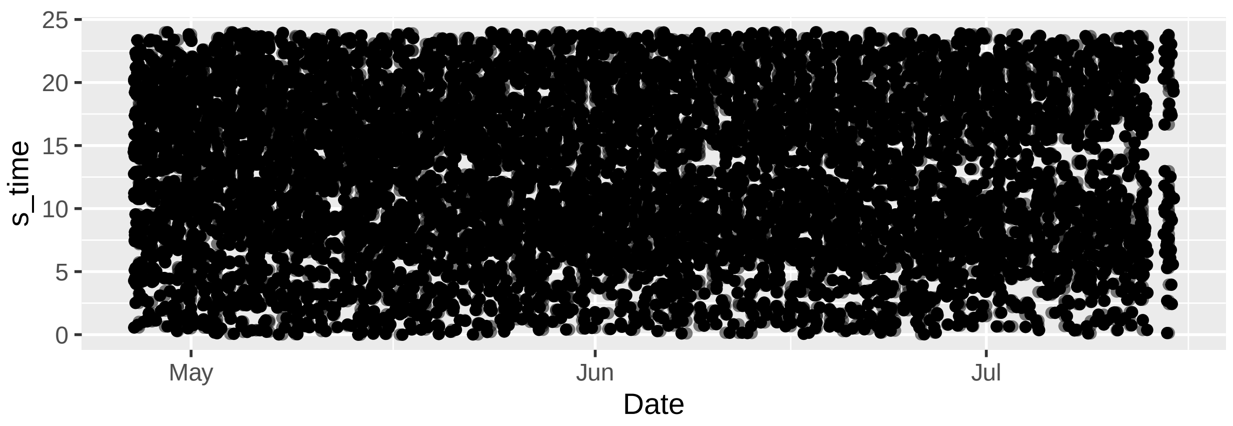


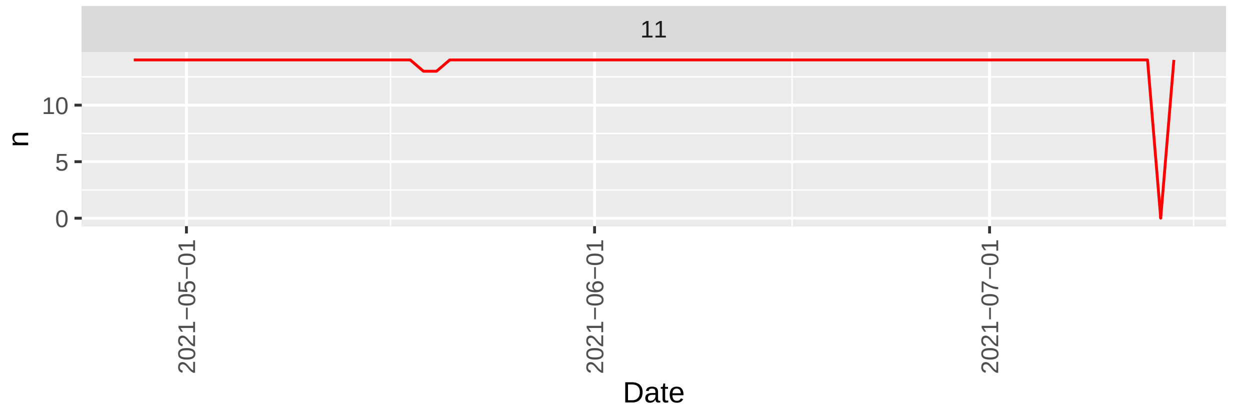
A birdseye view of all the data for a station

The `inspectDay` function can show the visits to a feeder for the whole period, in a single plot. It can also show a population plot similar to the previous section.

```
inspectDay(farmA_standard, thestation = 11)
```

```
## `summarise()` has grouped output by 'station'. You can override using the
## `.groups` argument.
## pad applied on the interval: day
```



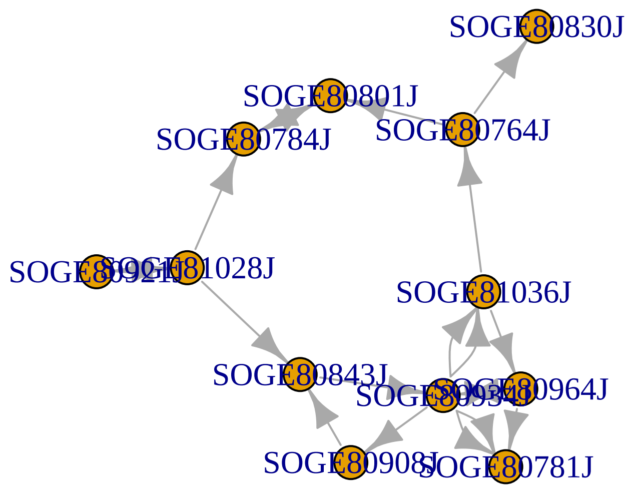


Building network visualizations and analyisis

Building the igraph objects and plotting

The following steps succesively converts the data into the network objects of the `igraph` package.

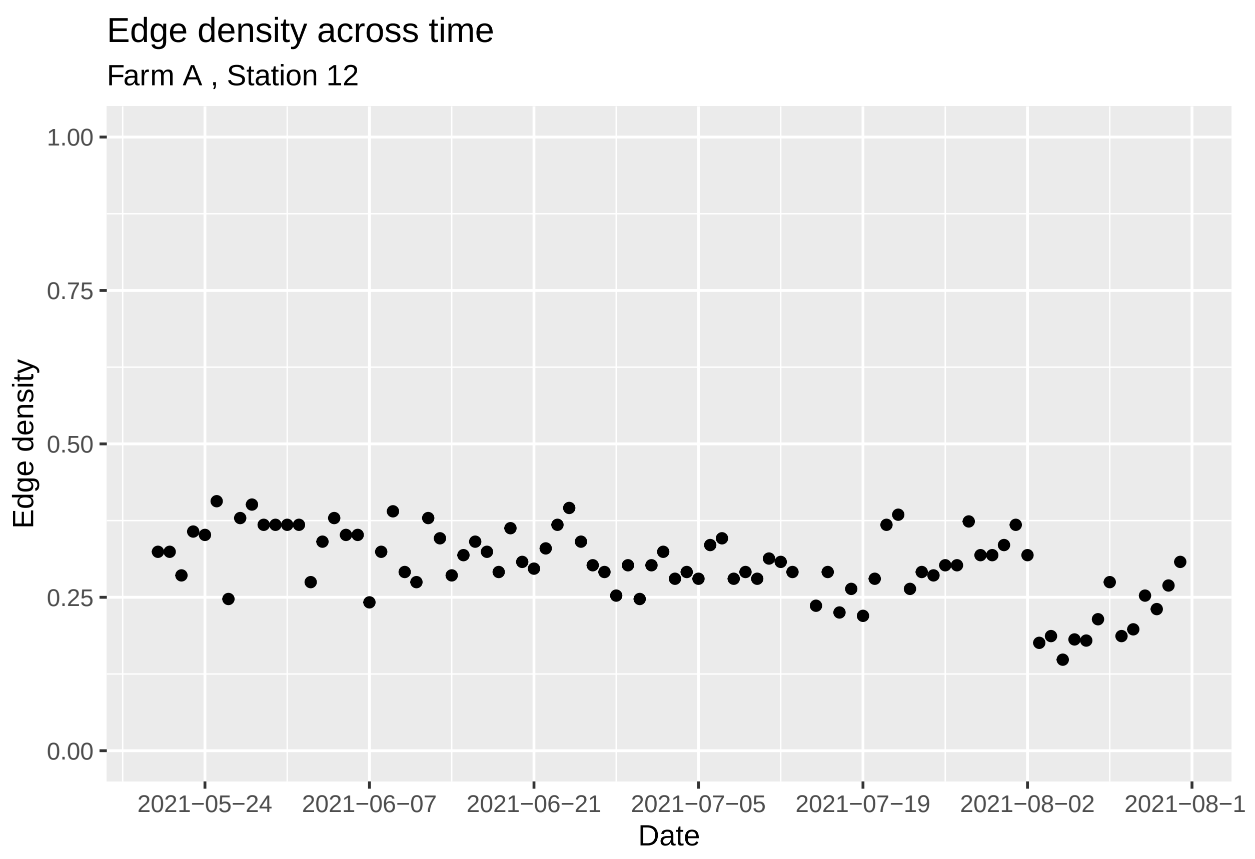
```
farmA_list <- makeAllStationsPerdate(farmA_standard, domerge = "F")

farmA_pairs <- makePairsPerStation(farmA_list, mythreshold = 5) # TAKES A LONG TIME

farmA_network <- makeIGraphObjects(farmA_pairs, directed = T)

plot(farmA_network[["12"]][["2021-05-20"]])
```



Making summarizations based on the network data

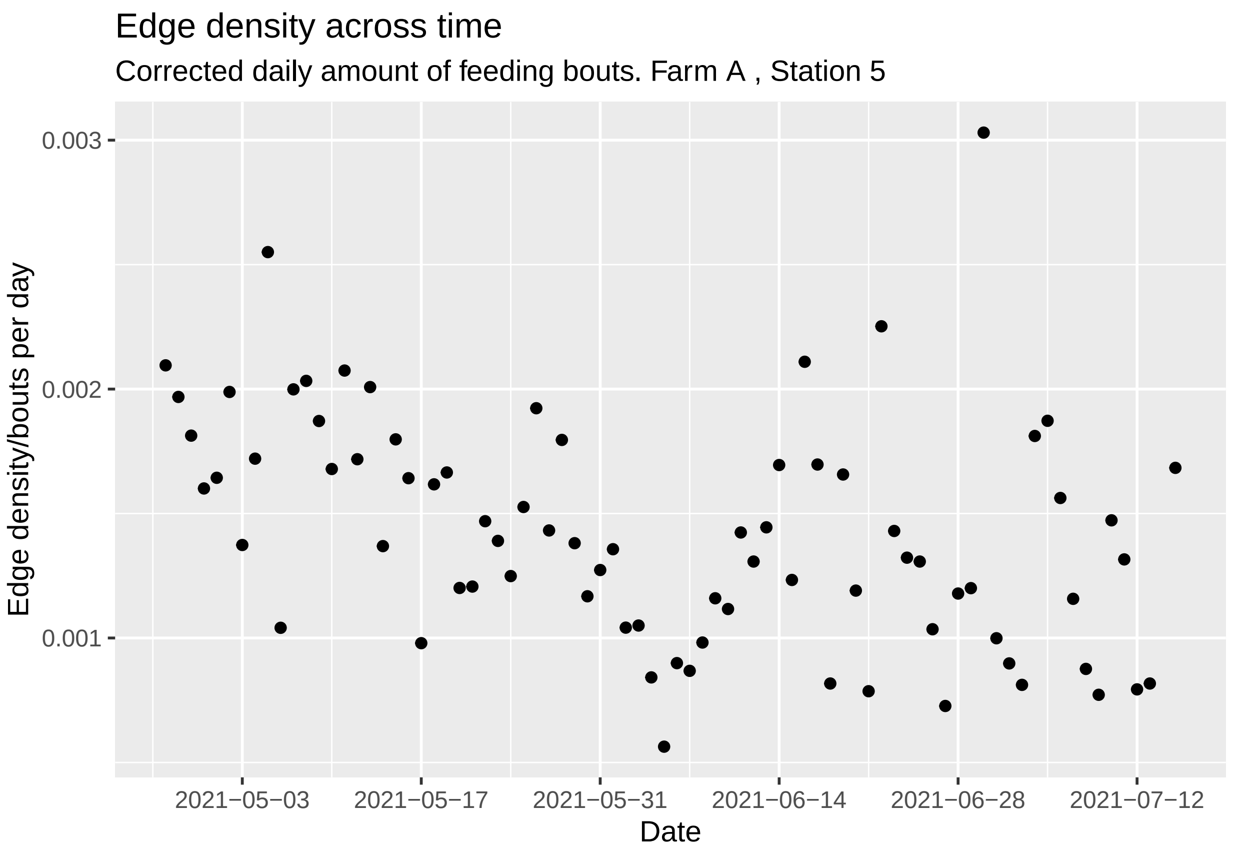
The following steps will analyze how a whole-network parameter, the Network Density progresses through time. It looks that there is a downward trend in the group we are studying.

```r
getmetheplot_pliz(site = "Farm A",
                  df = farmA_standard,
                  thestation = 12)
```



The reason why this trend occurs is not clear, but it could be that these animals are learning to avoid each other. Another possible explanation is that the animals are going less to the feeder as they grow, and thus there is less of a chance that the animals can interact with each other. The `getmetheplot_pliz_but_corrected_this_time()` function corrects the network density by the times the animals visit the feeder.

```
getmetheplot_pliz_but_corrected_this_time(site = "Farm A",
                                          df = farmA_standard,
                                          thestation = 5)
```



Even with this correction, we still see a downward trend.

Thanks for reading all of this, you can reach me at Microsoft Teams or e-mail at *lagog6@ulaval.ca*.