triact package for R: Analyzing the lying behavior of cows from accelerometer data

Michael Simmler, Stijn Brouwers

08.01.2023

This vignette contains executable examples for the intended use of the R package *triact*. With a few exceptions, the functionalities are presented with default parameters only. We recommend that you also read the help page to learn more about the many parameters that can be used to customize the behavior of the methods in *triact*. Furthermore, we detailed the background and rationale of the implemented analyses in the following publication:

> Simmler M., Brouwers S., 2023. *triact package for R: Analyzing the lying behavior of cows from accelerometer data*, in preparation

Setup

Since *triact* is intended for data with sampling frequency of >1 Hz, it is advisable to adjust R's global option `digits.secs`, which specifies the maximum number of digits to print when formatting time values in seconds (default is 0).

```
options(digits.secs = 3)
```

Via the global option `triact_table` the type of tables returned by *triact* can be specified. Options are `"data.frame"` (the default), `"tibble"`, and `"data.table"`.

```
options(triact_table = "data.frame")
```

Getting help

All functionalities of the *triact* R package are documented on the help page of the `Triact` R6 class.

```
?Triact
```

Inspecting the example data

The triact R package includes two raw data files from triaxial accelerometers (MSR145, MSR Electronics, Switzerland) attached to the left hind leg of dairy cows. The sampling frequency was 5 Hz. Each file represents one day of recording of one cow.

```
dir <-  system.file("extdata", package = "triact")

files <- list.files(dir)

print(files)
```

```
## [1] "cow01_5hz.csv" "cow02_5hz.csv"
```

Inspecting one of the files reveals a file header and the semicolon-separated data starting after the line with `"*Data`". This is an example of how files imported by *triact* might look. However, *triact* can handle any kind of delimiter-separated text files.

```
cat(paste(readLines(file.path(dir, files[1]), n = 30), collapse = "\n"))
```

```
## *CREATOR
## msr_cutter.exe;[V6.06.02]
## msr2csv.exe;[V6.06.02]
##
## *STARTTIME
## 2021-06-29;06:00:00;
##
## *MODUL
## NAME;MSR314553;MSR314553;MSR314553
## *NAME
## NAME;6A;6A;6A
##
## ID;[C26113 V5.66];[C26113 V5.66];[C26113 V5.66]
##
## *CHANNEL
## TIME;ACC x;ACC y;ACC z
##
## *UNIT
## ;G;G;G
##
## *LIMITS
## ALARM;;;
## RECRD;;;
## LIMIT1;;;
## LIMIT2;;;
##
## *DATA
## 2021-06-29 06:00:00.055;-0.048;1.032;-0.063
## 2021-06-29 06:00:00.258;-0.048;1.000;-0.063
## 2021-06-29 06:00:00.461;-0.048;1.000;-0.063
```

Importing data

The typical *triact* workflow starts by creating a new object of the `Triact` class.

```
my_triact <- Triact$new()
```

Acceleration data is then imported into the `Triact` object (here named 'my_triact'). Using the `$load_files()` method the data from raw data files can be imported. This method has many arguments, which allow adapting the behavior to read the specific format of the delimiter-separated data file. Important for correct processing of the files are `id_substring` and `timeFwdUpRight_cols`. The first specifies the substring of the file names that uniquely identifies the cows, while the latter maps the columns as found in the file to the *time*, *forward* acceleration, *up* acceleration, and *right* acceleration as understood by *triact*. Notes: Alternatively, you can read your files with your own routine and then use the `$load_table` method to import a data.frame into the `Triact` object. If you suspect that you have accidentally placed the accelerometers rotated by 180° in the sagittal plane, you can get help with `$check_orientation()`.

```
my_triact$load_files(input = dir,
                     id_substring = c(1, 5),
                     timeFwdUpRight_cols = c(1, 2 ,3, 4),
                     skip = "*DATA")
```

Imported raw data and added analyses can be accessed between any step of the workflow via the `$data` field.

```
head(my_triact$data)
```

```
##      id                time acc_fwd acc_up acc_right
## 1 cow01 2021-06-29 06:00:00  -0.048  1.032    -0.063
## 2 cow01 2021-06-29 06:00:00  -0.048  1.000    -0.063
## 3 cow01 2021-06-29 06:00:00  -0.048  1.000    -0.063
## 4 cow01 2021-06-29 06:00:00  -0.048  1.032    -0.063
## 5 cow01 2021-06-29 06:00:00  -0.048  1.032    -0.031
## 6 cow01 2021-06-29 06:00:01  -0.079  1.032    -0.031
```

```
str(my_triact$data)
```

```
## 'data.frame':    354462 obs. of  5 variables:
##  $ id       : Factor w/ 2 levels "cow01","cow02": 1 1 1 1 1 1 1 1 1 1 ...
##  $ time     : POSIXct, format: "2021-06-29 06:00:00.055" "2021-06-29 06:00:00.257" ...
##  $ acc_fwd  : num  -0.048 -0.048 -0.048 -0.048 -0.048 -0.079 -0.048 -0.048 -0.079 -0.079 ...
##  $ acc_up   : num  1.03 1 1 1.03 1.03 ...
##  $ acc_right: num  -0.063 -0.063 -0.063 -0.063 -0.031 -0.031 -0.031 -0.031 -0.031 -0.031 ...
```

Adding analyses

Calling add_... methods triggers analyses of lying behavior and the calculation of proxies for the level of physical activity. These analyses are obtained for each time point of your acceleration data and added in a new column to the tabular data in the `Triact` object.

Detecting standing and lying posture

The `$add_lying()` method performs the classification into lying and standing. The results are (silently) added to the data in the `Triact` object as a column named *lying*.

```
my_triact$add_lying()
```

Detecting lying laterality

The `$add_side()` method performs the determination of lying laterality (left/right lying side). The results are (silently) added to the data in the `Triact` object as a column named *side*. Crucial for correct determination of the lying side is the correct specification of the hind leg the accelerometer was mounted on (parameter `left_leg = TRUE` for left, or `FALSE` for right).

```
my_triact$add_side(left_leg = TRUE)
```

Calculating proxies for the physical activity level

The `$add_activity()` method performs the calculation of proxies for the physical activity level of the cow(s). By default, the L2 norm of the vector of the dynamic body acceleration is calculated, and 'adjusted' to a value of zero during lying bouts (AdjL2DBA). See `?Triact` and Simmler & Brouwers (2023) for more information on the available proxies for the physical activity level.

```
my_triact$add_activity()
```

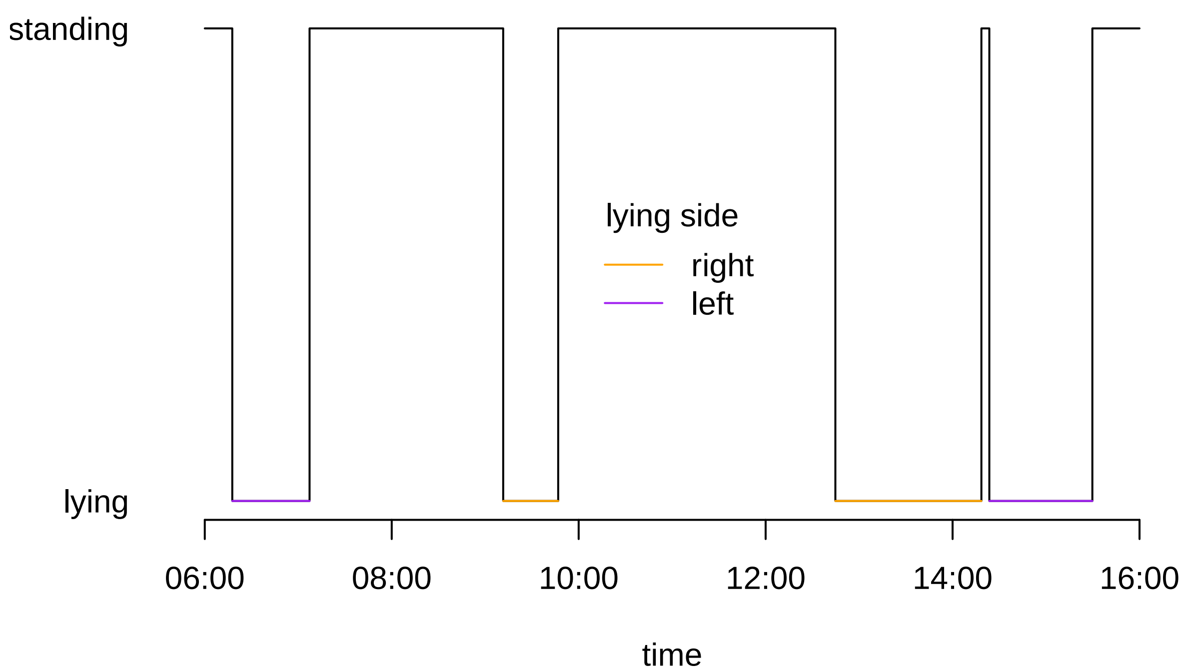
The data in the `Triact` object now additionally contains the added analyses (last three columns).

```
str(my_triact$data)
```

```
## 'data.frame':    354462 obs. of  9 variables:
##  $ id       : Factor w/ 2 levels "cow01","cow02": 1 1 1 1 1 1 1 1 1 1 ...
##  $ time     : POSIXct, format: "2021-06-29 06:00:00.055" "2021-06-29 06:00:00.257" ...
##  $ acc_fwd  : num  -0.048 -0.048 -0.048 -0.048 -0.048 -0.079 -0.048 -0.048 -0.079 -0.079 ...
##  $ acc_up   : num  1.03 1 1 1.03 1.03 ...
##  $ acc_right: num  -0.063 -0.063 -0.063 -0.063 -0.031 -0.031 -0.031 -0.031 -0.031 -0.031 ...
##  $ bout_nr  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ lying    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ side     : Factor w/ 2 levels "L","R": NA NA NA NA NA NA NA NA NA NA ...
##  $ AdjL2DBA : num  0.0453 0.032 0.032 0.0453 0.032 ...
##  - attr(*, "index")= int(0)
##   ..- attr(*, "__lying")= int [1:354462] 1 2 3 4 5 6 7 8 9 10 ...
```

The *triact* package does not come with visualization capabilities. But the data can easily be accessed and plotted with base R or packages dedicated to graphics (e.g. ggplot2). The following example shows how to access the data of a single cow (here with ID 013) and to visualize the lying behavior.

```
cow_id = "cow01"

data_013 <- my_triact$data[my_triact$data$id == cow_id, ]

plot(!lying ~ time, data = data_013,
     type = "l", ylab = "", yaxt = "n", bty = "n")

lines(ifelse(side == "R", 0, NA) ~ time, data = data_013, col = "orange")
lines(ifelse(side == "L", 0, NA) ~ time, data = data_013, col = "purple")

axis(2, at = c(0, 1),
     labels = c("lying", "standing"),
     las = 1,
     lwd = 0)

legend(x = "center",
       legend = c("right", "left"),
       col = c("orange", "purple"),
       lwd = 1, bty = "n", title = "lying side")
```



Summarizing results

Using the $summarize… methods the analyses added to the `Triact` object is summarized per time period, which represent either the standing/lying bouts or regular intervals, e.g. 1 h or 24 h.

Summarizing per lying/standing bout

With `$summarize_bouts()` a summary is created for the individual lying and standing bouts, with duration, mean activity, and lying side (for a lying bout). In the output we see that the first bout per cow is not completely observed (`startTime` is missing) and therefor NAs are returned for measures such as duration. See parameter `calc_for_incomplete`in case you want to change this behavior.

```
bouts_summary <- my_triact$summarize_bouts()
```

```
head(bouts_summary)
```

```
##      id bout_nr           startTime             endTime  duration lying side
## 1 cow01       1                <NA> 2021-06-29 06:17:40        NA FALSE <NA>
## 2 cow01       2 2021-06-29 06:17:40 2021-06-29 07:07:16  49.59968  TRUE    L
## 3 cow01       3 2021-06-29 07:07:16 2021-06-29 09:11:33 124.28197 FALSE <NA>
## 4 cow01       4 2021-06-29 09:11:33 2021-06-29 09:46:51  35.30985  TRUE    R
## 5 cow01       5 2021-06-29 09:46:51 2021-06-29 12:44:45 177.90022 FALSE <NA>
## 6 cow01       6 2021-06-29 12:44:45 2021-06-29 14:18:29  93.73198  TRUE    R
##   meanAdjL2DBA
## 1           NA
## 2   0.00000000
## 3   0.20761060
## 4   0.00000000
## 5   0.08440727
## 6   0.00000000
```

If only the lying bouts are of interest, the `bout_type` parameter can be specified accordingly.

```
bouts_summary <- my_triact$summarize_bouts(bout_type = "lying")
```

```
head(bouts_summary)
```

```
##      id bout_nr           startTime             endTime  duration lying side
## 1 cow01       2 2021-06-29 06:17:40 2021-06-29 07:07:16  49.59968  TRUE    L
## 2 cow01       4 2021-06-29 09:11:33 2021-06-29 09:46:51  35.30985  TRUE    R
## 3 cow01       6 2021-06-29 12:44:45 2021-06-29 14:18:29  93.73198  TRUE    R
## 4 cow01       8 2021-06-29 14:23:35                <NA>        NA  TRUE    L
## 5 cow02       1                <NA> 2021-06-29 07:07:26        NA  TRUE    R
## 6 cow02       3 2021-06-29 12:56:43 2021-06-29 14:38:01 101.29500  TRUE    R
##   meanAdjL2DBA
## 1            0
## 2            0
## 3            0
## 4           NA
## 5           NA
## 6            0
```

Summarizing per regular intervals

With `$summarize_intervals()` the summary is obtained per regular intervals, by default per hour.

```
int_summary <- my_triact$summarize_intervals()
```

The NAs in the output are a result of incompletely observed intervals (first and last interval of each cow) and of dependencies of measures on information of incompletely observed bouts (can also affect second and further intervals). See parameter `calc_for_incomplete` in case you want to change this behavior. The NaN on the other hand do not indicate missing information: For example, if the cow was not standing in the interval, the mean activity during standing is not zero, but cannot be calculated (thus NaN, "not a number").

```
head(int_summary)
```

```
##      id           startTime          centerTime             endTime duration
## 1 cow01 2021-06-29 06:00:00 2021-06-29 06:30:00 2021-06-29 07:00:00       NA
## 2 cow01 2021-06-29 07:00:00 2021-06-29 07:30:00 2021-06-29 08:00:00 59.99968
## 3 cow01 2021-06-29 08:00:00 2021-06-29 08:30:00 2021-06-29 09:00:00 59.99970
## 4 cow01 2021-06-29 09:00:00 2021-06-29 09:30:00 2021-06-29 10:00:00 60.00307
## 5 cow01 2021-06-29 10:00:00 2021-06-29 10:30:00 2021-06-29 11:00:00 59.99970
## 6 cow01 2021-06-29 11:00:00 2021-06-29 11:30:00 2021-06-29 12:00:00 59.99968
##   durationStanding durationLying durationLyingLeft durationLyingRight
## 1               NA            NA                NA                 NA
## 2         52.72782      7.271868          7.271868            0.00000
## 3         59.99970      0.000000          0.000000            0.00000
## 4         24.69321     35.309862          0.000000           35.30986
## 5         59.99970      0.000000          0.000000            0.00000
## 6         59.99968      0.000000          0.000000            0.00000
##   meanAdjL2DBA meanAdjL2DBALying meanAdjL2DBAStanding
## 1           NA                NA                  NA
## 2   0.23734730                 0          0.27008066
## 3   0.16785237               NaN          0.16785237
## 4   0.04852252                 0          0.11790694
## 5   0.10169885               NaN          0.10169885
## 6   0.06039537               NaN          0.06039537
```

```
str(int_summary)
```

```
## 'data.frame':    20 obs. of  12 variables:
##  $ id                 : Factor w/ 2 levels "cow01","cow02": 1 1 1 1 1 1 1 1 1 1 ...
##  $ startTime          : POSIXct, format: "2021-06-29 06:00:00" "2021-06-29 07:00:00" ...
##  $ centerTime         : POSIXct, format: "2021-06-29 06:30:00" "2021-06-29 07:30:00" ...
##  $ endTime            : POSIXct, format: "2021-06-29 07:00:00" "2021-06-29 08:00:00" ...
##  $ duration           : num  NA 60 60 60 60 ...
##  $ durationStanding   : num  NA 52.7 60 24.7 60 ...
##  $ durationLying      : num  NA 7.27 0 35.31 0 ...
##  $ durationLyingLeft  : num  NA 7.27 0 0 0 ...
##  $ durationLyingRight : num  NA 0 0 35.3 0 ...
##  $ meanAdjL2DBA       : num  NA 0.2373 0.1679 0.0485 0.1017 ...
##  $ meanAdjL2DBALying  : num  NA 0 NaN 0 NaN NaN 0 0 0 NA ...
##  $ meanAdjL2DBAStanding: num  NA 0.27 0.168 0.118 0.102 ...
##  - attr(*, "index")= int(0)
##   ..- attr(*, "__startTime")= int [1:20] 1 11 2 12 3 13 4 14 5 15 ...
```

The intervals can be specified quite flexibly: In case of 30 min intervals and starting 10 min after the full hour we can specify `interval` and `lag_in_s` parameters accordingly.

```
int_summary <- my_triact$summarize_intervals(interval = "30 min",
                                             lag_in_s = 10 * 60)
```

```
head(int_summary)
```

```
##      id           startTime          centerTime             endTime duration
## 1 cow01 2021-06-29 05:40:00 2021-06-29 05:55:00 2021-06-29 06:10:00       NA
## 2 cow01 2021-06-29 06:10:00 2021-06-29 06:25:00 2021-06-29 06:40:00 30.00152
## 3 cow01 2021-06-29 06:40:00 2021-06-29 06:55:00 2021-06-29 07:10:00 29.99812
## 4 cow01 2021-06-29 07:10:00 2021-06-29 07:25:00 2021-06-29 07:40:00 30.00152
## 5 cow01 2021-06-29 07:40:00 2021-06-29 07:55:00 2021-06-29 08:10:00 29.99813
## 6 cow01 2021-06-29 08:10:00 2021-06-29 08:25:00 2021-06-29 08:40:00 30.00150
##   durationStanding durationLying durationLyingLeft durationLyingRight
## 1               NA            NA                NA                 NA
## 2         7.671342      22.33017          22.33017                  0
## 3         2.728640      27.26948          27.26948                  0
## 4        30.001517       0.00000           0.00000                  0
## 5        29.998133       0.00000           0.00000                  0
## 6        30.001500       0.00000           0.00000                  0
##   meanAdjL2DBA meanAdjL2DBALying meanAdjL2DBAStanding
## 1           NA                NA                  NA
## 2   0.03410266                 0           0.1333706
## 3   0.05575711                 0           0.6129823
## 4   0.27305669               NaN           0.2730567
## 5   0.20804566               NaN           0.2080457
## 6   0.17560383               NaN           0.1756038
```

```
str(int_summary)
```

```
## 'data.frame':    42 obs. of  12 variables:
##  $ id                 : Factor w/ 2 levels "cow01","cow02": 1 1 1 1 1 1 1 1 1 1 ...
##  $ startTime          : POSIXct, format: "2021-06-29 05:40:00" "2021-06-29 06:10:00" ...
##  $ centerTime         : POSIXct, format: "2021-06-29 05:55:00" "2021-06-29 06:25:00" ...
##  $ endTime            : POSIXct, format: "2021-06-29 06:10:00" "2021-06-29 06:40:00" ...
##  $ duration           : num  NA 30 30 30 30 ...
##  $ durationStanding   : num  NA 7.67 2.73 30 30 ...
##  $ durationLying      : num  NA 22.3 27.3 0 0 ...
##  $ durationLyingLeft  : num  NA 22.3 27.3 0 0 ...
##  $ durationLyingRight : num  NA 0 0 0 0 ...
##  $ meanAdjL2DBA       : num  NA 0.0341 0.0558 0.2731 0.208 ...
##  $ meanAdjL2DBALying  : num  NA 0 0 NaN NaN NaN NaN 0 0 NaN ...
##  $ meanAdjL2DBAStanding: num  NA 0.133 0.613 0.273 0.208 ...
##  - attr(*, "index")= int(0)
##   ..- attr(*, "__startTime")= int [1:42] 1 22 2 23 3 24 4 25 5 26 ...
```

With `bouts == TRUE` we can request that, additionally, the bouts within the intervals are summarized. For measures such as the number of lying bouts or mean lying bout duration, a weighted mean is calculated with the weights being the proportion of the individual bout overlapping with the respective interval. With `side = TRUE` we can additionally request a differentiation of all results by lying side.

```
int_summary <- my_triact$summarize_intervals(bouts = TRUE,
                                             side = TRUE)
```

```
str(int_summary)
```

```
## 'data.frame':    20 obs. of  22 variables:
##  $ id                         : Factor w/ 2 levels "cow01","cow02": 1 1 1 1 1 1 1 1 1 1 ...
##  $ startTime                  : POSIXct, format: "2021-06-29 06:00:00" "2021-06-29 07:00:00" ...
##  $ centerTime                 : POSIXct, format: "2021-06-29 06:30:00" "2021-06-29 07:30:00" ...
##  $ endTime                    : POSIXct, format: "2021-06-29 07:00:00" "2021-06-29 08:00:00" ...
##  $ duration                   : num  NA 60 60 60 60 ...
##  $ durationStanding           : num  NA 52.7 60 24.7 60 ...
##  $ durationLying              : num  NA 7.27 0 35.31 0 ...
##  $ durationLyingLeft          : num  NA 7.27 0 0 0 ...
##  $ durationLyingRight         : num  NA 0 0 35.3 0 ...
##  $ meanAdjL2DBA               : num  NA 0.2373 0.1679 0.0485 0.1017 ...
##  $ meanAdjL2DBALying          : num  NA 0 NaN 0 NaN NaN 0 0 0 NA ...
##  $ meanAdjL2DBAStanding       : num  NA 0.27 0.168 0.118 0.102 ...
##  $ meanAdjL2DBALyingLeft      : num  NA 0 NaN NaN NaN NaN NaN NaN 0 NA ...
##  $ meanAdjL2DBALyingRight     : num  NA NaN NaN 0 NaN NaN 0 0 0 NA ...
##  $ nBoutsStanding             : num  NA 0.424 0.483 0.167 0.337 ...
##  $ nBoutsLying                : num  NA 0.147 0 1 0 ...
##  $ nBoutsLyingLeft            : num  NA 0.147 0 0 0 ...
##  $ nBoutsLyingRight           : num  NA 0 0 1 0 ...
##  $ wMeanDurationStandingBout  : num  NA 124 124 148 178 ...
##  $ wMeanDurationLyingBout     : num  NA 49.6 NaN 35.3 NaN ...
##  $ wMeanDurationLyingBoutLeft : num  NA 49.6 NaN NaN NaN ...
##  $ wMeanDurationLyingBoutRight: num  NA NaN NaN 35.3 NaN ...
##  - attr(*, "index")= int(0)
##   ..- attr(*, "__startTime")= int [1:20] 1 11 2 12 3 13 4 14 5 15 ...
```

Extracting posture transitions

Using `$extract_liedown()` and `$extract_standup()`, the raw acceleration data (and added analyses) of the posture transitions, i.e., lying-to-standing and standing-to-lying, can be extracted.

With default parameters, only the time of the transition, bout nr of the lying bout, and lying side (if available) is returned.

```
st_ups <- my_triact$extract_standup()
```

```
print(st_ups)
```

```
##      id                time bout_nr side
## 1 cow01 2021-06-29 07:07:16       2    L
## 2 cow01 2021-06-29 09:46:51       4    R
## 3 cow01 2021-06-29 14:18:29       6    R
## 4 cow01 2021-06-29 15:29:43       8    L
## 5 cow02 2021-06-29 07:07:26       1    R
## 6 cow02 2021-06-29 14:38:00       3    R
## 7 cow02 2021-06-29 15:43:00       5    L
```

When specifying 'sec_before' and 'sec_after', time series around the exact moment of posture transition as detected by *triact* are returned. The result is a list with tables (one table per posture transition).

```
l_downs <- my_triact$extract_liedown(sec_before = 3, sec_after = 3)
```

```
head(l_downs[[1]])
```

```
##      id                time acc_fwd acc_up acc_right bout_nr lying side
## 1 cow01 2021-06-29 06:17:37  -0.238  0.968    -0.031       1 FALSE <NA>
## 2 cow01 2021-06-29 06:17:37  -0.175  0.968    -0.031       1 FALSE <NA>
## 3 cow01 2021-06-29 06:17:37  -0.175  1.000     0.000       1 FALSE <NA>
## 4 cow01 2021-06-29 06:17:38  -0.143  1.000    -0.063       1 FALSE <NA>
## 5 cow01 2021-06-29 06:17:38  -0.143  1.000    -0.063       1 FALSE <NA>
## 6 cow01 2021-06-29 06:17:38  -0.111  1.032    -0.031       1 FALSE <NA>
##     AdjL2DBA
## 1 0.09929753
## 2 0.09897980
## 3 0.13270268
## 4 0.09400532
## 5 0.09400532
## 6 0.14817895
```