telemR Guide

last updated March 12 2020

telemR is an R package written to import, plot, analyze, and export telemetry data.

Any bugs or feature requests should be made at https://github.com/jevanveen/telemR.

Required packages

telemR uses functions found in the tidyverse package, including lubridate (which is not automatically installed with tidyverse). Also, "cowplot" is nice for putting together graphs, but not strictly necessary.

To get started, install the packages:

```
install.packages("devtools")
install.packages("tidyverse")
install.packages("lubridate")
install.packages("cowplot")
devtools::install_github("jevanveen/telemR")
```

Then load these packages

```
library(tidyverse)
library(lubridate)
library(cowplot)
library(telemR)
```

Importing data

Telemetry data come in large matrices which are not particularly easy to manipulate in R. The first two functions in telemR read telemetry data and place them in "tidy format." More info about tidy data: https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html.

Currently, telemR supports importing data obtained by Starr life sciences and Oddi-Star devices.

To import Starr files, they **must be saved as raw .asc files in "new format" with date and time in leftmost column.** If you have a lot of old files saved in a different format, I could change the function to recognize differently formatted data.

Before importing data, **you must define meta data** in a data frame before import. The option `meta_data_group` is where you add grouping information from your experiment, and is required. You must use the **same mouse designations** you used in the telemetry software as the function looks for these names/numbers in your .asc file. An example meta data table looks like:

```
my.meta.data <- tibble("TMX" = c("54_2934", "52_3065", "48_3061", "50_3063"),
                       "VEH" = c("53_3066", "55_2936", "49_3062", "51_3064"))
```

Now to import data, use the function `read_starr`:

```
tmx.telem <- read_starr(raw_asc = "~/Box/Tamoxifen Paper/zz157jev.asc",
                        meta_data_group = my.meta.data)
```

by default, the function removes time points with NA values. Note that it removes these time points entirely (for all mice) to avoid problems in statistics downstream. If things look screwy, you can always turn off NA removal by changing that option:

```
tmx.telem <- read_starr(raw_asc = "~/Box/Tamoxifen Paper/zz157jev.asc",
                        meta_data_group = my.meta.data, trim_na = F)
```

only meta_data_group is *required* but there are two other meta data fields which can be supplied: `meta_data_sex` and `meta_data_xover`. `meta_data_sex` is where you can stash the sexes of animals if you need them later. `meta_data_xover` is required if you have done a crossover drug treatment that will need uncrossing later:

```
sex <- tibble("F" = c("54_2934", "52_3065", "53_3066", "55_2936"),
              "M" = c("48_3061", "50_3063","49_3062", "51_3064"))
xovers <- tibble("A" = c("54_2934", "52_3065", "53_3066", "55_2936"),
                 "B" = c("48_3061", "50_3063","49_3062", "51_3064"))

tmx.telem <- read_starr("~/Box/Tamoxifen Paper/zz157jev.asc",
                        meta_data_group = my.meta.data, meta_data_xover = xovers,
                        meta_data_sex = sex)
```

```
## Parsed with column specification:
## cols(
##   `48_3061 ZZ Deg. C YY/MM/DD HH:MM:SS` = col_character(),
##   `48_3061 ZZ Deg. C Data` = col_double(),
##   `48_3061 ZZ Cnts Data` = col_double(),
##   `49_3062 ZZ Deg. C Data` = col_double(),
##   `49_3062 ZZ Cnts Data` = col_double(),
##   `50_3063 ZZ Deg. C Data` = col_double(),
##   `50_3063 ZZ Cnts Data` = col_double(),
##   `51_3064 ZZ Deg. C Data` = col_double(),
##   `51_3064 ZZ Cnts Data` = col_double(),
##   `52_3065 ZZ Deg. C Data` = col_double(),
##   `52_3065 ZZ Cnts Data` = col_double(),
##   `53_3066 ZZ Deg. C Data` = col_double(),
##   `53_3066 ZZ Cnts Data` = col_double(),
##   `54_2934 ZZ Deg. C Data` = col_double(),
##   `54_2934 ZZ Cnts Data` = col_double(),
##   `55_2936 ZZ Deg. C Data` = col_double(),
##   `55_2936 ZZ Cnts Data` = col_double()
## )
```

```
## Parsed with column specification:
## cols(
##   `Experiment Logfile: logfile157.log` = col_character()
## )
```

If all has gone right, you should see `tmx.telem` in your global environment. You can open it and examine that the data look correct by clicking on it in the environment pane or just typing it's name in the console:

```
tmx.telem
```

```
## # A tibble: 44,008 x 7
##     DegC Counts Time                Mouse   Group Xover Sex
##    <dbl>  <dbl> <dttm>              <fct>   <fct> <fct> <fct>
##  1 NaN        0 2020-01-04 11:45:00 48_3061 TMX   B     M
##  2  37.4    228 2020-01-04 11:50:00 48_3061 TMX   B     M
##  3  37.4    223 2020-01-04 11:55:00 48_3061 TMX   B     M
##  4  37.4    251 2020-01-04 12:00:00 48_3061 TMX   B     M
##  5  37.4    266 2020-01-04 12:05:00 48_3061 TMX   B     M
##  6  37.4     99 2020-01-04 12:10:00 48_3061 TMX   B     M
##  7  37.0     37 2020-01-04 12:15:00 48_3061 TMX   B     M
##  8  37.0     56 2020-01-04 12:20:00 48_3061 TMX   B     M
##  9  36.7     46 2020-01-04 12:25:00 48_3061 TMX   B     M
## 10  36.7      0 2020-01-04 12:30:00 48_3061 TMX   B     M
## # ... with 43,998 more rows
```

The process is very similar for importing files produced by Oddi telemetry probes. The main difference is that the Oddi software saves one xlsx file per probe. Name each xlsx file something that's easy to remember, and put them all into one folder with nothing else in it. **Use the excel filenames (without .xlsx) in your meta data assignments.** Then import with `read_oddi`. Because the Oddi transmitters are often affixed to tails, the `read_oddi` function by default looks for, and tries to drop, detached probes. It does this by looking for probes that read < 23c for > 4 hours. Once that condition is met, all future data are deleted.

```
xovers <- tibble("A" = c("52t", "53t", "54t", "55t"),
                 "B" = c("48t", "49t", "50t", "51t"))

meta.data <- tibble("TMX" = c("48t", "50t", "52t", "54t"),
                    "VEH" = c("49t", "51t", "53t", "55t"))

sex <- tibble("F" = c("52t", "53t", "54t", "55t",
                      "48t", "49t", "50t", "51t"))

tmx.oddi <- read_oddi("~/Dropbox/JEV/Zhi single cell paper/telemetry/",
                      meta_data_group = meta.data)
```

both of the read functions produce a data frame, which is passed to other funcitons as the `tidy_telem` variable. For Starr produced files, movement data are stored under the name "Counts" (`tidy_telem$Counts`) and temperature data are stored under the name "DegC" (`tidy_telem$DegC`)

Exporting data

Now if you want to get back out of R and make graphs or do analysis in Prism or Excel, you can make an easily readable .xlsx file with the function `export_telem`:

```
export_telem(tidy_telem = tmx.telem, filename = "~/Desktop/myfilename.xlsx")
```

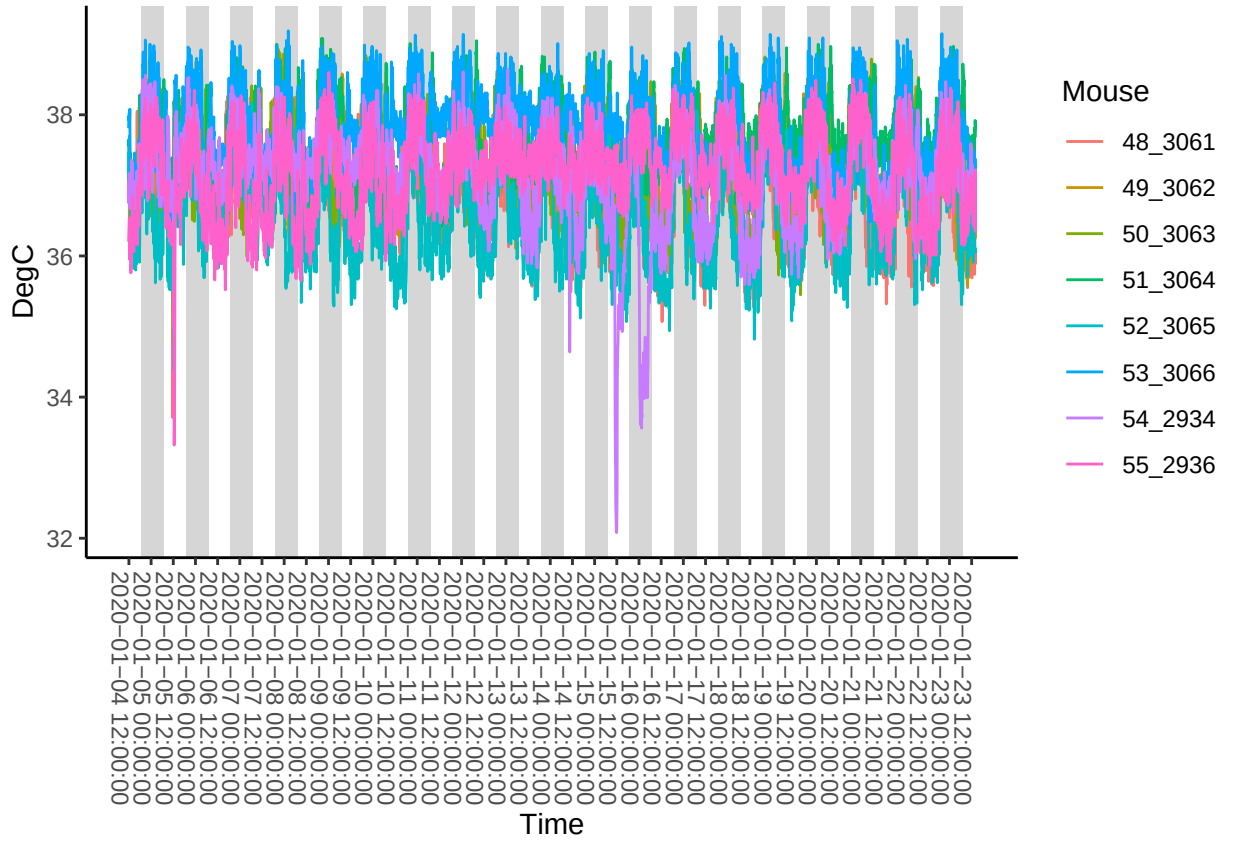
Graphing data

But if you want to stay in R, there are some nice graphing and analysis options. First, basic graphing with `graph_telem`

```
graph_telem(tidy_telem = tmx.telem)
```

```
## Warning: Removed 43 rows containing non-finite values (stat_summary).
```

```
## Warning: Removed 43 rows containing non-finite values (stat_summary).
```

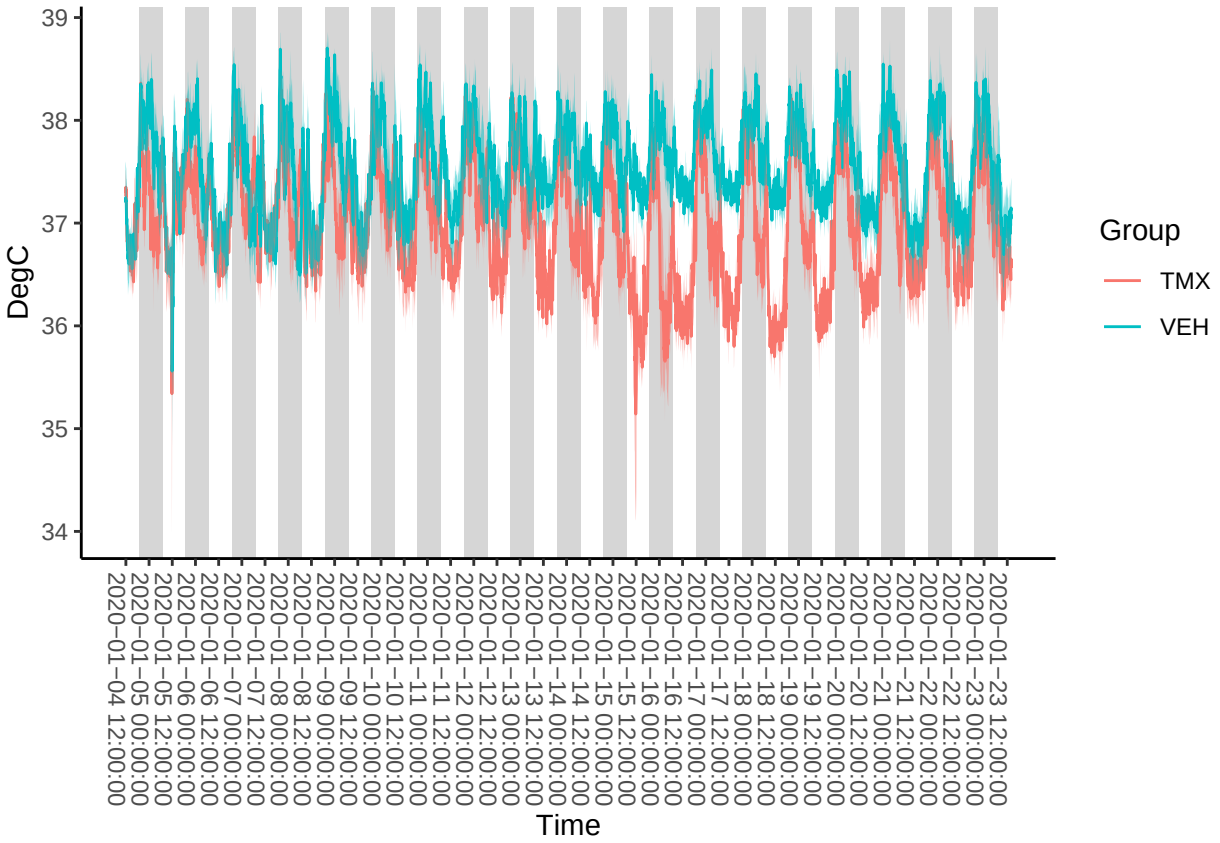


The default is to graph all mice individually so that you can see if there are any screwy data, but this is usually noisy, and takes a long time because of the thousands of data points. Usually more useful are grouped data, or data collapsed by day:

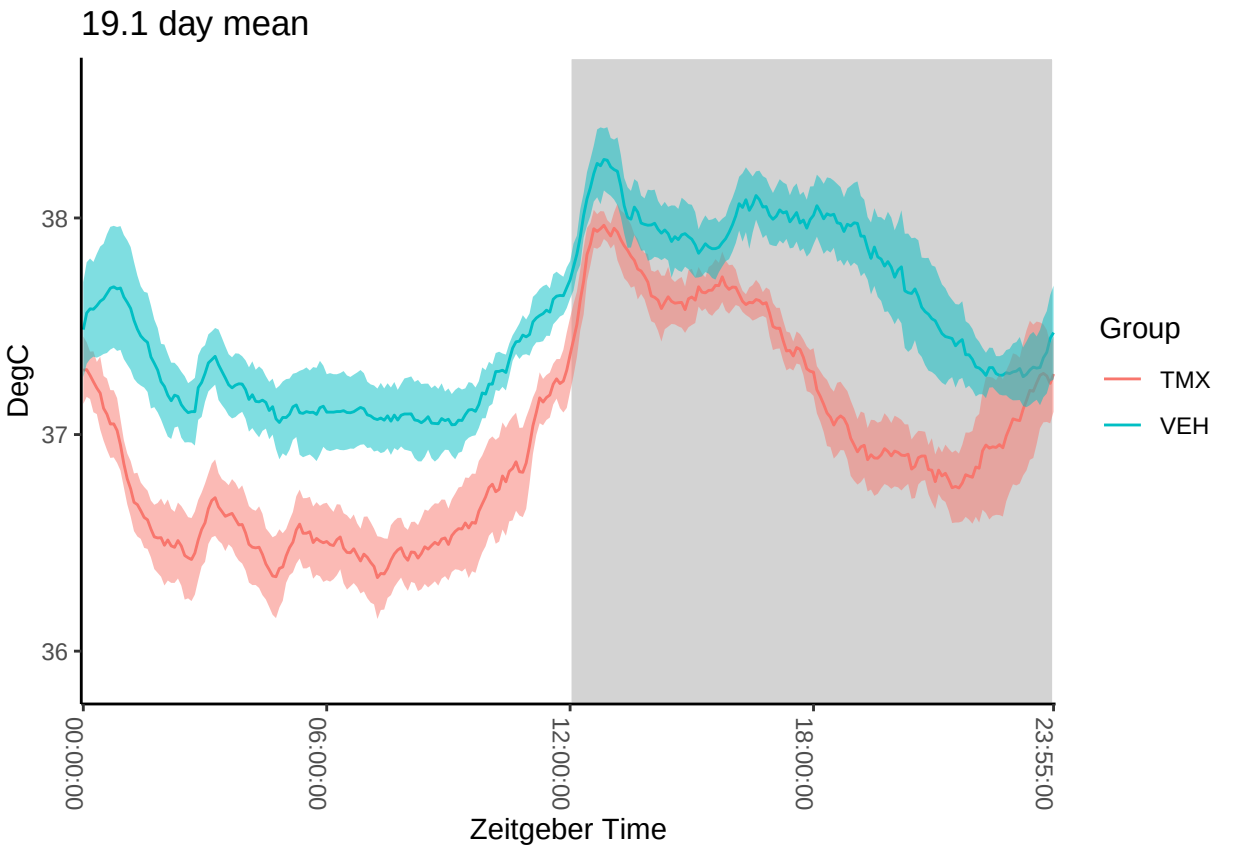
```
graph_telem(tidy_telem = tmx.telem, group_by = "Group")
```

```
## Warning: Removed 43 rows containing non-finite values (stat_summary).
```

```
## Warning: Removed 43 rows containing non-finite values (stat_summary).
```




```
graph_telem(tidy_telem = tmx.telem, group_by = "Group", one_day_avg = T)
```



Now... your life will be much easier and more organized if you learn to use the "pipe operator" in R (%>%). It passes whatever is before it as the first argument of the following function, so

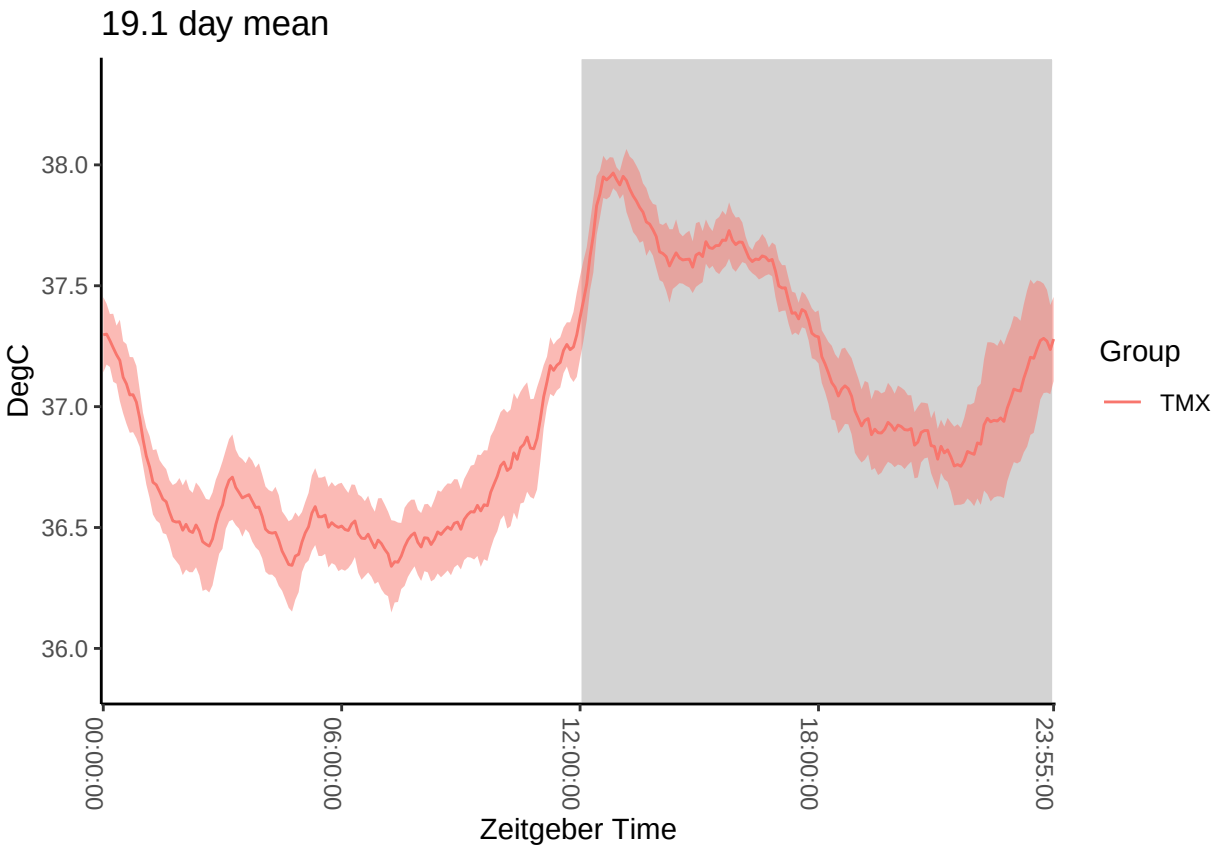
```
graph_telem(tidy_telem = tmx.telem, group_by = "Group", one_day_avg = T)
```

is the same as

```
tmx.telem %>% graph_telem(group_by = "Group", one_day_avg = T)
```

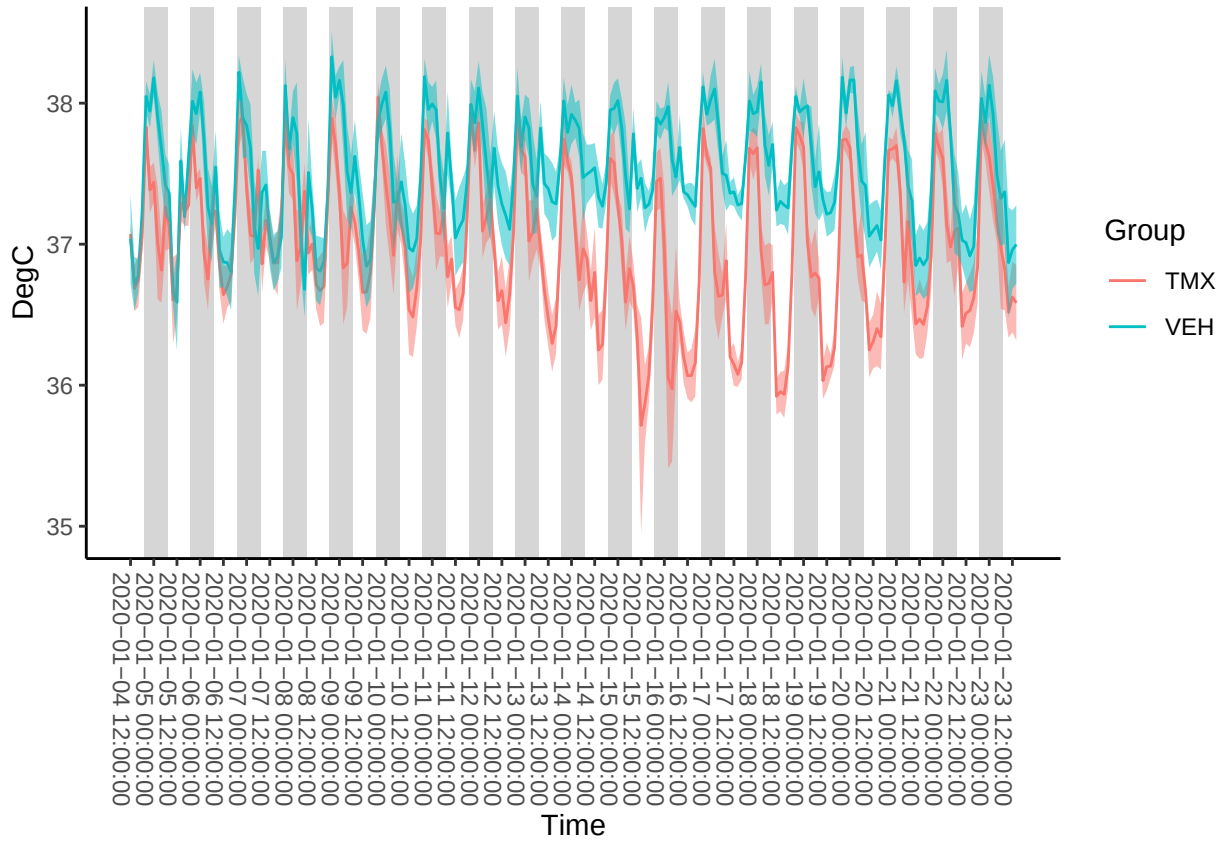
This really cleans things up as we restructure the data on multiple levels. For example, we might want to trim out some treatment groups, some mice or some times, before graphing, which we can do with trim_telem. Piping just makes it look neater:

```
tmx.telem %>%
  trim_telem(keep_groups = "TMX") %>%
  graph_telem(group_by = "Group", one_day_avg = T)
```



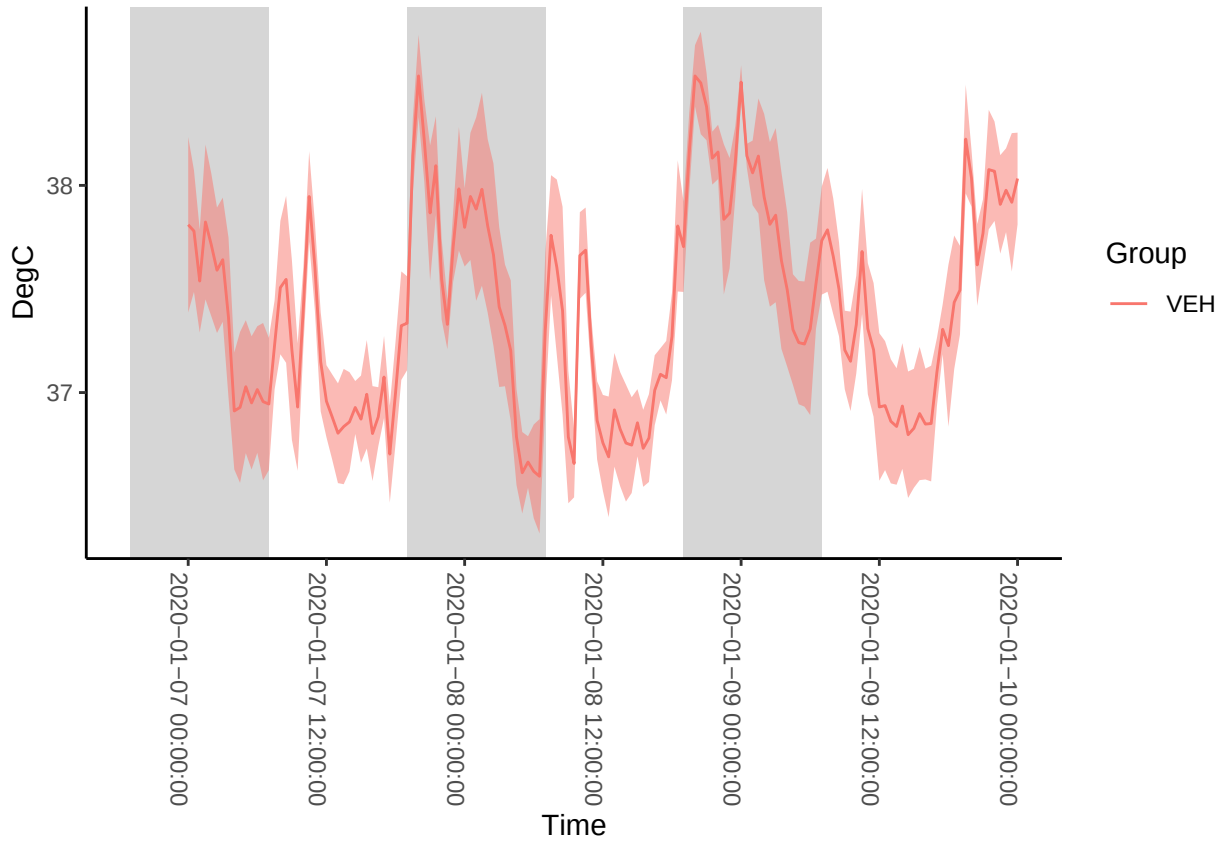
Often telemetry data are too noisy to make sense of because of the 5 minute sampling interval. We can average over user specified times with the function `fuzz_telem`:

```
tmx.telem %>%
  fuzz_telem("2 hours") %>%
  graph_telem(group_by = "Group")
```



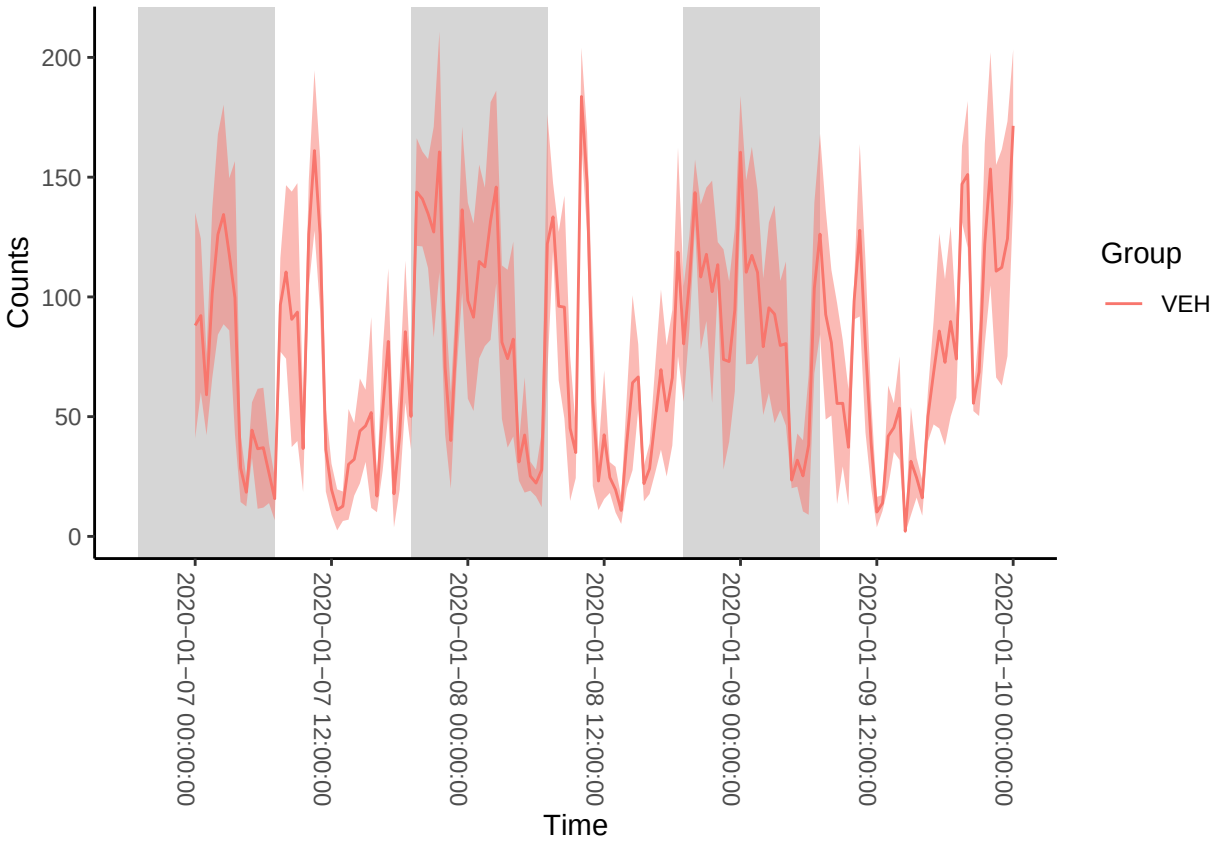
It's important to note that these functions can be layered on top of each other, so if you want to see data for just a few days, that are fuzzed, for only VEH mice, you'd write

```
tmx.telem %>%
  trim_telem(start_time = "2020-01-07 00:00:00", end_time = "2020-01-10 00:00:00") %>%
  fuzz_telem("30 minutes") %>%
  trim_telem(keep_groups = "VEH") %>%
  graph_telem(group_by = "Group")
```



All of these analyses have been done for temperature by default, but if your original data includes counts, those graphs can be made by specifying telem_var as `"Counts"` in graph_telem. Everything else can stay the same:

```
tmx.telem %>%
  trim_telem(start_time = "2020-01-07 00:00:00", end_time = "2020-01-10 00:00:00") %>%
  fuzz_telem("30 minutes") %>%
  trim_telem(keep_groups = "VEH") %>%
  graph_telem(group_by = "Group", telem_var = "Counts")
```



Statistical analyses

To test for significant differences between groups, we use the function `lme_telem`, which performs a linear mixed effects analysis:

```
my.anova <- lme_telem(tidy_telem = tmx.telem, telem_var = "DegC")
```

This produces a lot of data including residuals and other factors in the linear model. For a summary of the test, just type:

```
my.anova$AOV
```

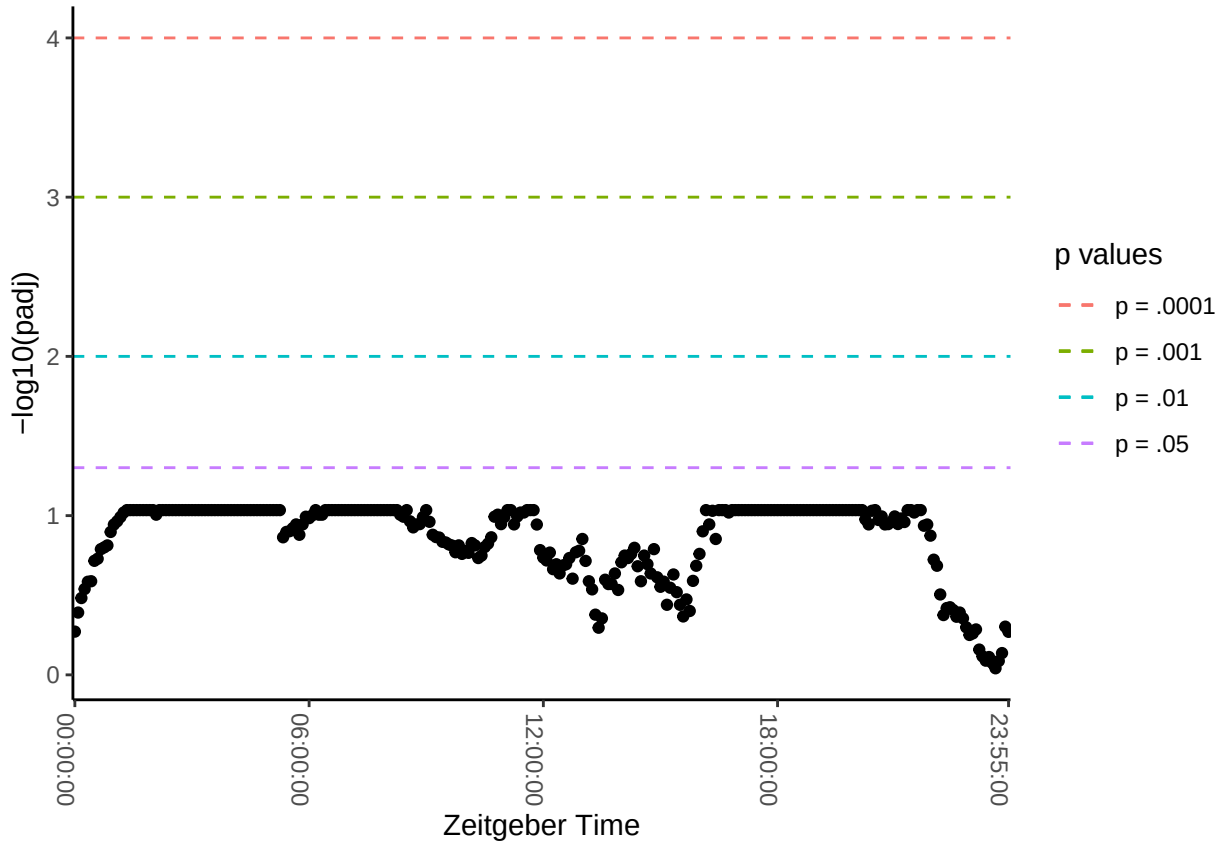
```
##            Model  df       AIC      BIC     logLik   Test  L.Ratio p-value
## baseline       1   4 2932.6144 2955.584 -1462.3072
## Group          2   5 2927.6791 2956.391 -1458.8395 1 vs 2    6.935  0.0085
## Time           3 292 -168.2457 1508.536   376.1228 2 vs 3 3669.925  <.0001
## GroupXTime     4 579 -364.1575 2960.693   761.0787 3 vs 4  769.912  <.0001
```

Note that the LME model is prone to failure if there is too much missing data, as is often common in telemetry files. Because of this, the `lme_telem` function collpases everything to one day by default, which usually solves the problem. Note that `lme_telem` is also compatible with the other functions, and piping, so if you want to do the test for a specific window, you could write:

```
my.anova <- tmx.telem %>%
  trim_telem(start_time = "2020-01-07 00:00:00", end_time = "2020-01-10 00:00:00") %>%
  lme_telem()
```

Finally, there is a function for doing multiple comparisons called `multcomp_telem`. It will either return a table with FDR corrected p values at each time point, or a graphical snapshot. This function requires a formula, which is unquoted, telling it which comparisons to make such as: compare temp between groups: `DegC ~ Group`. Or compare movement between sexes: `Counts ~ Sex`.

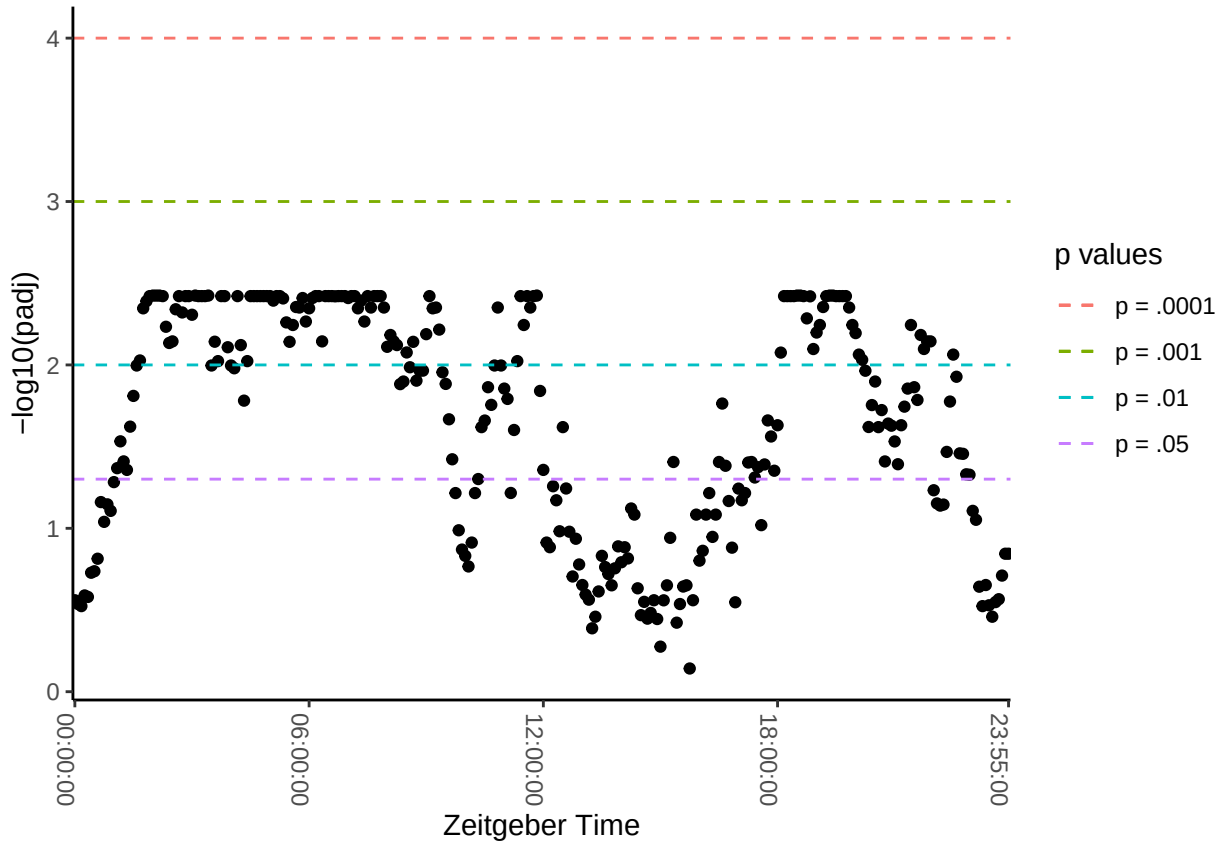
```
multcomp_telem(tidy_telem = tmx.telem, DegC ~ Group, plot_or_table = "plot")
```



In this graph, nothing reaches significance.

Again, the high sampling rate of telemetry data can be an issue for multiple corrections, because each addtional t-test decreases overall power. To combat this, by default the data are first collpased to a one day snapshot, but you can also use the functions you used in graphing. For example with these data, I can see that we observe TMX effects most strongly from the 16th to the 20th, so if I write:

```
tmx.telem %>%
  trim_telem(start_time = "2020-01-16 00:00:00", end_time = "2020-01-20 00:00:00") %>%
  multcomp_telem(DegC ~ Group, plot_or_table = "plot")
```



Now, things look much more significant.

And if we want to see or save the full stats table, we can just change that option:

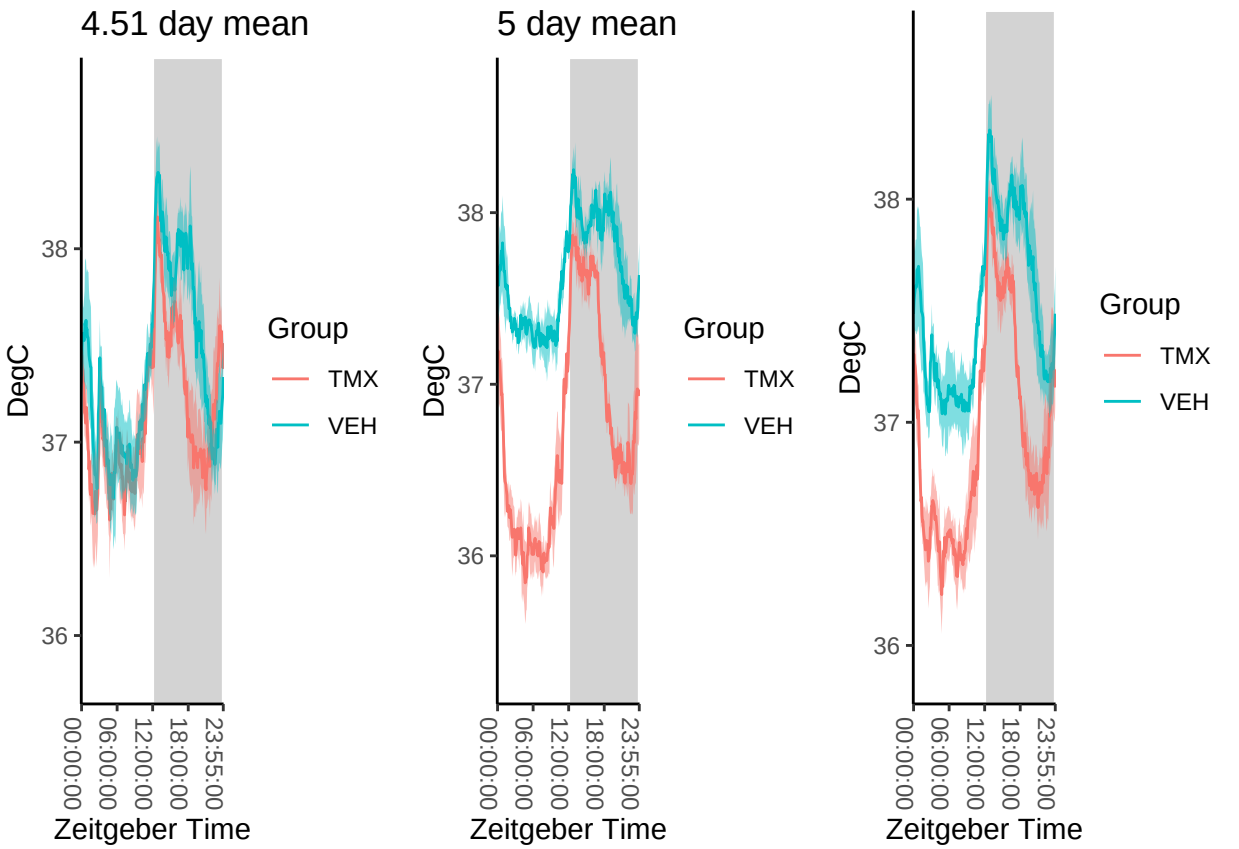
```
tmx.telem %>%
  trim_telem(start_time = "2020-01-16 00:00:00", end_time = "2020-01-20 00:00:00") %>%
  multcomp_telem(DegC ~ Group, plot_or_table = "table")
```

```
## # A tibble: 288 x 13
##    Time  estimate estimate1 estimate2 statistic p.value parameter conf.low
##    <fct>    <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>
##  1 00:0~   -0.492      37.1      37.6     -1.25  0.257       6.00    -1.45
##  2 00:0~   -0.471      37.1      37.6     -1.20  0.277       6.00    -1.43
##  3 00:1~   -0.472      37.1      37.6     -1.17  0.289       5.68    -1.47
##  4 00:1~   -0.554      37.1      37.6     -1.32  0.235       5.79    -1.59
##  5 00:2~   -0.526      37.1      37.7     -1.30  0.241       5.85    -1.52
##  6 00:2~   -0.551      37.1      37.7     -1.60  0.162       5.99    -1.40
##  7 00:3~   -0.531      37.1      37.6     -1.62  0.158       5.97    -1.34
##  8 00:3~   -0.644      37.0      37.7     -1.76  0.128       6.00    -1.54
##  9 00:4~   -0.794      37.0      37.8     -2.48  0.0497      5.76    -1.59
## 10 00:4~   -0.746      37.0      37.7     -2.23  0.0697      5.70    -1.58
## # ... with 278 more rows, and 5 more variables: conf.high <dbl>, method <chr>,
## #   alternative <chr>, padj <dbl>, stat_summary <chr>
```

Combining/Aligning experiments

There are two options for combining experiments: `combine_telem` which collpases each experiment to a one day snapshot, then combines them, and `align_telem` which changes the dates on one experiment. First, let's look at `combine telem` which is fairly trivial. For this example, we can make two separate tidy_telem files from 'tmx.telem' by using `trim_telem`:

```
t1 <- tmx.telem %>%
  trim_telem(start_time = "2020-01-04 00:00:00", end_time = "2020-01-09 00:00:00")

t2 <- tmx.telem %>%
  trim_telem(start_time = "2020-01-15 00:00:00", end_time = "2020-01-20 00:00:00")

p1 <- t1 %>% graph_telem(group_by = "Group", one_day_avg = T)

p2 <- t2 %>% graph_telem(group_by = "Group", one_day_avg = T)

p3 <- combine_telem(t1, t2) %>%
  graph_telem(group_by = "Group")

plot_grid(p1, p2, p3, ncol = 3)
```

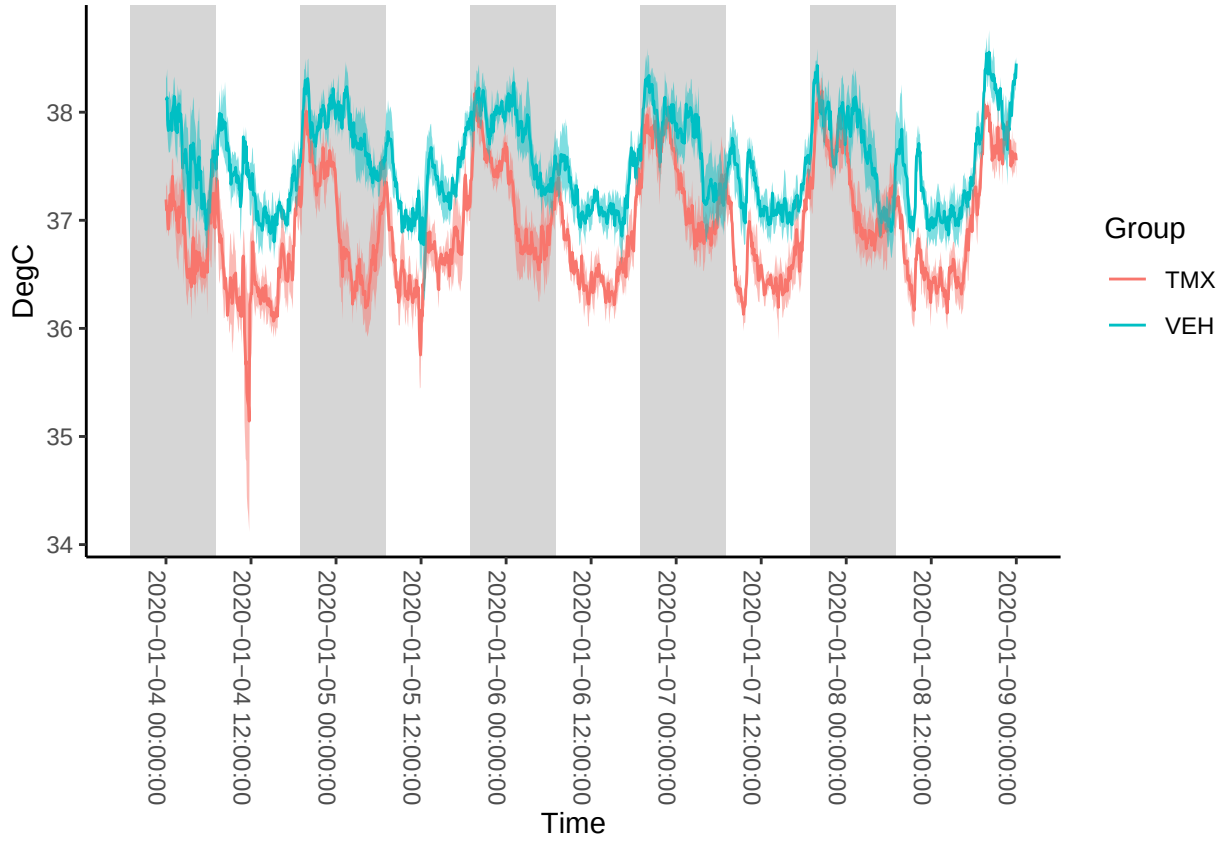


As you can see, the combined experiment is an average of the two experiments, since the mouse and metadata designations are all the same.

If we want to combine experiments without collpasing first, we use the fucntion `align_telem`. By definition, to align two telemetry files, one has to decide how the dates and times should be aligned. By default,

`align_telem` will align the first dates in the two tidy_telem tibbles, but you can also specify which dates to align. You cannot currently align different times (i.e. align 12:00 on day1 with 4:00 on day2). As with `combine_telem`, data for mice with the same number and metadata are averaged, so if you need them separate, they need to be named differently, or have different meta data.

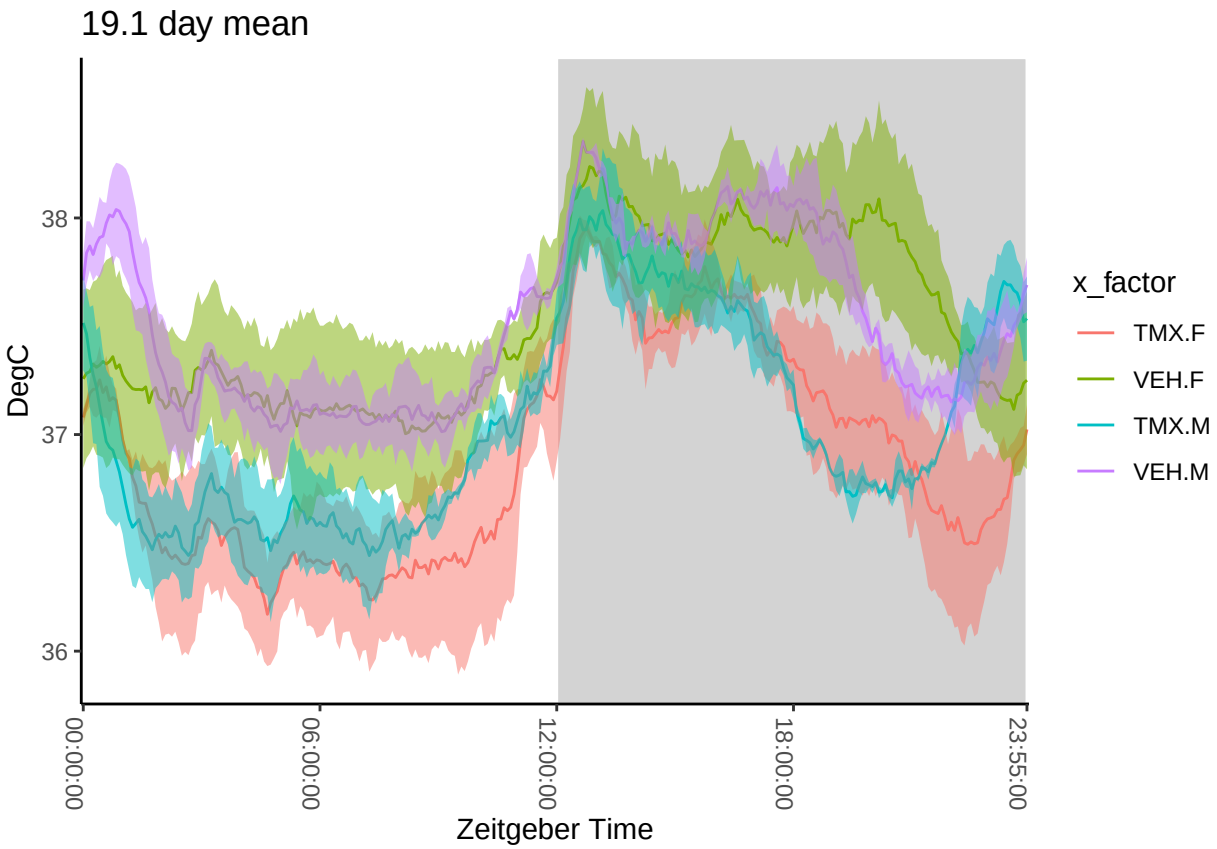
```
align_telem(t1, t2, align_date_1 = "2020-01-04", align_date_2 = "2020-01-15") %>%
  graph_telem(group_by = "Group")
```



Multiplying Factors

As we've learned there are currently three slots for inputting meta data, `meta_data_group`, which is required, as well as `meta_data_sex` and `meta_data_xover`. Often we might want to see a graph that displays two levels of meta data, like treatment and sex for example. We can create a new factor that is the cross product of two existing factors, on the fly, by using `x_factor_telem`. This will produce a new cross-factor, named "x_factor", which is the interaction of two meta data factors that you are interested in. To see it, you have to tell `graph_telem` to `group_by = "x_factor"`:

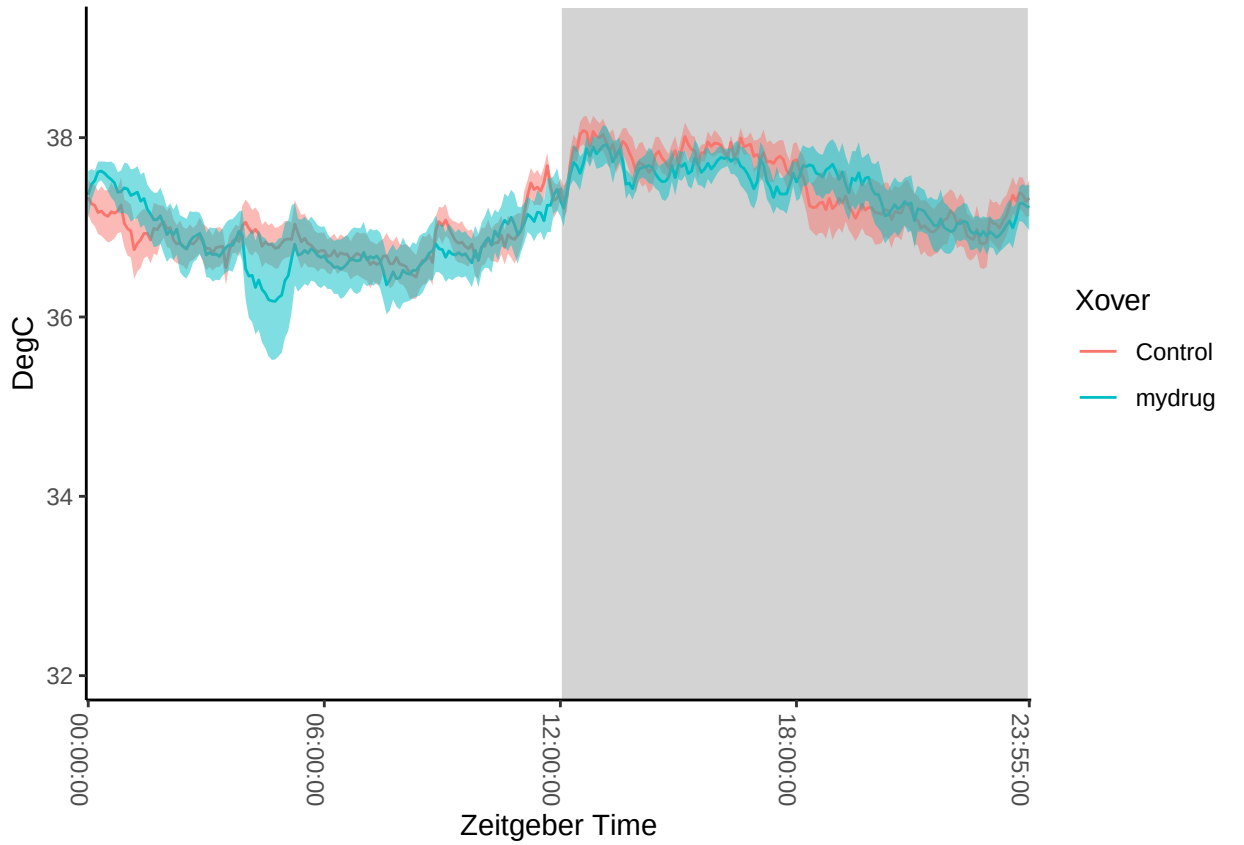
```
tmx.telem %>%
  x_factor_telem(telem_var_1 = "Group", telem_var_2 = "Sex") %>%
  graph_telem(group_by = "x_factor", one_day_avg = T)
```



Crossover analyses

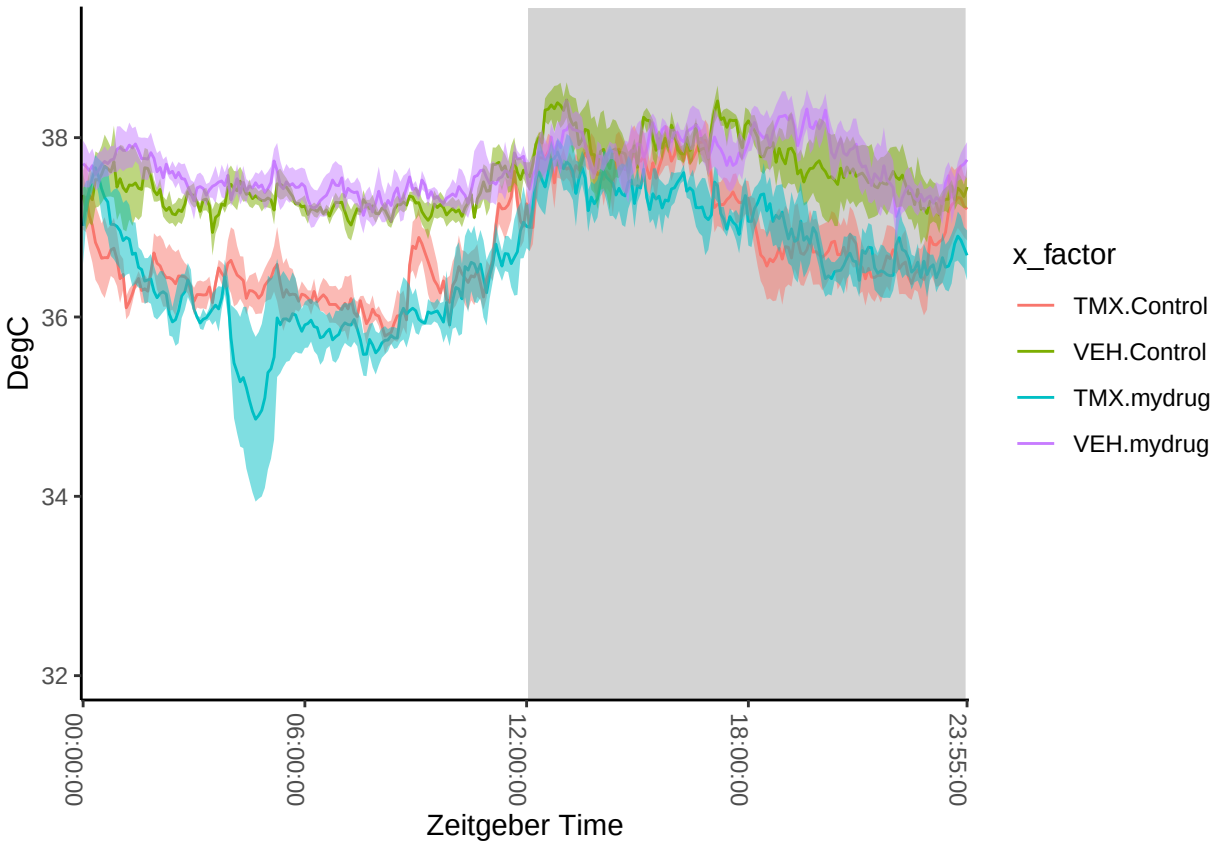
Often we setup an experiment in a balanced, crossover design where half of the animals are given a treatment on one day, while the other half are given control treatment. Then on a second day, the treatment groups are switched. We can demultiplex this kind of experiment with `x_over_telem`. For this function to work, you must have divided all samples into two groups by supplying meta data to `meta_data_xover`. The function will assume that you named these meta data groups "A" and "B", but if you named it something else, you just have to tell the function by defining `xover_pattern_1` and `xover_pattern_2`. The function returns a collapsed telemetry file. To see the unmixed data, you have to tell `graph_telem` to `group_by = "Xover"`:

```
tmx.telem %>%
  x_over_telem(day_1_start = "2020-01-15 00:00:00",
               day_2_start = "2020-01-17 00:00:00", xover_var = "mydrug") %>%
  graph_telem(group_by = "Xover", telem_var = "DegC")
```



And if you still want to maintain your original treatment groups in your crossover unmixing, you can do that by combining `x_factor_telem` with `x_over_telem`. Note though, that this is one of the few cases where the order of functions applies matters, as you must unmix with `x_over_telem` before making your cross factor with `x_factor_telem`:

```
tmx.telem %>%
  x_over_telem(day_1_start = "2020-01-15 00:00:00",
               day_2_start = "2020-01-17 00:00:00", xover_var = "mydrug") %>%
  x_factor_telem("Group", "Xover") %>%
  graph_telem(group_by = "x_factor", one_day_avg = T)
```


38
36
34
32
DegC
00:00:00
06:00:00
12:00:00
18:00:00
23:55:00
Zeitgeber Time
x_factor
TMX.Control
VEH.Control
TMX.mydrug
VEH.mydrug