The Seatrack database and R

A short intro

Jens Åström

NINA
Norsk institutt for naturforskning

23 Feb, 2019

Database description

- PostgreSQL (with PostGIS) database hosted at NINA: seatrack.nina.no
  - Only available from Polar institute and NINA's IP-range. So use VPN when travelling
- Can be accessed with "standard tools" like PgAdmin, ODBC, Access, and R
- This presentation only covers working with the database through R

R-package “seatrackR”

- Custom functions for working with the database
- Code at: https://github.com/NINAnor/seatrack-db/tree/master/seatrackR

Install:

```
devtools::install_github("NINAnor/seatrack-db/seatrackR",
  build_vignettes = True
)
```

- When in trouble, update the package first and restart R. If the problem persists, notify jens.astrom@nina.no.

Connecting

- Note that you need a personal user name
- There are three types of users
  - seatrack_reader (only reads, most users)
  - seatrack_writer (can write logger logistict, position data, and upload files to archive)
  - seatrack_metadata_writer (can alter lookup-tables)

```
require(seatrackR)
connectSeatrack(
  Username = "testreader",
  Password = "testreader"
)
```

Connecting

Remember to change your default password!

Don’t use a sensitive password, e.g. something you use on another important places. I can’t swear that noone will be able to see it!

```
changeSeatrackPassword(password = "hunter2")
```

Querying the database

- The `connectSeatrack` function creates a `DBI` connection called `con`. You can use this with the `DBI`, `dplyr`, `sf` packages.
- An example of using R's ordinary functions, reading an entire table:

```
loggers <- dbReadTable(con, Id(schema = "loggers", table = "logger_info"))
head(loggers)
                                     id logger_id logger_serial_no logger_model
1 59aabc6a-e64a-11e8-b4a0-005056b165f3         1              C101       mk4083
2 59abba0c-e64a-11e8-b4a0-005056b165f3         2              C102       mk4083
3 59ac0c00-e64a-11e8-b4a0-005056b165f3         3              C103       mk4083
4 59ac5dfe-e64a-11e8-b4a0-005056b165f3         4              C104       mk4083
5 59acad9a-e64a-11e8-b4a0-005056b165f3         5              C105       mk4083
6 59acfe9e-e64a-11e8-b4a0-005056b165f3         6              C772       mk4083
  producer production_year  project
1 Biotrack            2015 SEATRACK
2 Biotrack            2015 SEATRACK
3 Biotrack            2015 SEATRACK
4 Biotrack            2015 SEATRACK
5 Biotrack            2015 SEATRACK
6 Biotrack            2015 SEATRACK
```

Querying the database

Writing custom SQL queries in the standard R way.

```
LOTEKLoggersQ <- "SELECT * FROM loggers.logger_info WHERE producer = 'LOTEK'"

LOTEKLoggers <- dbGetQuery(con, LOTEKLoggersQ)
head(LOTEKLoggers)
                                     id logger_id logger_serial_no logger_model
1 3d5c8132-e64b-11e8-b4a0-005056b165f3      1266        L280-0664          LAT
2 69e32a60-e657-11e8-b4a0-005056b165f3      3652             1107          LAT
3 69e44710-e657-11e8-b4a0-005056b165f3      3653             1158          LAT
4 69e54304-e657-11e8-b4a0-005056b165f3      3654             1161          LAT
5 69e63854-e657-11e8-b4a0-005056b165f3      3655             1181          LAT
6 69e72fe8-e657-11e8-b4a0-005056b165f3      3656             1183          LAT
  producer production_year project
1    LOTEK            2017  SEAPOP
2    LOTEK            2009  SEAPOP
3    LOTEK            2009  SEAPOP
4    LOTEK            2009  SEAPOP
5    LOTEK            2009  SEAPOP
6    LOTEK            2009  SEAPOP
```

Querying the database

Simple operations like this could also be done using dplyr/dbplyr. The filtering here actually happens on the database side, but you can specify it using dplyr commands in R.

```
BASLoggers <- dbReadTable(con, Id(schema = "loggers", table = "logger_info")) %>%
  filter(producer == "BAS")

head(BASLoggers)
                                     id logger_id logger_serial_no logger_model
1 a508fbc2-e64a-11e8-b4a0-005056b165f3       238              8974         mk13
2 a509a2c0-e64a-11e8-b4a0-005056b165f3       239              8987         mk13
3 a509d682-e64a-11e8-b4a0-005056b165f3       240              8986         mk13
4 af23786c-e64a-11e8-b4a0-005056b165f3       241           18B387         mk18
5 af23deec-e64a-11e8-b4a0-005056b165f3       242           18B406         mk18
6 af24152e-e64a-11e8-b4a0-005056b165f3       243           18B395         mk18
  producer production_year project
1      BAS            2009  SEAPOP
2      BAS            2009  SEAPOP
3      BAS            2009  SEAPOP
4      BAS            2011  SEAPOP
5      BAS            2011  SEAPOP
6      BAS            2011  SEAPOP
```

Querying the database

Dplyr can also do joins using "dbplyr". A silly example:

```
require(dbplyr)
status <- tbl(con, in_schema("individuals", "individ_status"))
loggers <- tbl(con, in_schema("loggers", "logger_info"))

loggerEggs <- status %>%
  inner_join(loggers, by = c("logger_id" = "logger_id")) %>%
  group_by(producer) %>%
  filter(!is.na(eggs)) %>%
  select(
    producer,
    eggs
  )
```


NINA

Querying the database

```
ggplot(loggerEggs, aes(eggs, producer)) +
  geom_boxplot()
```

NINA

Querying the database

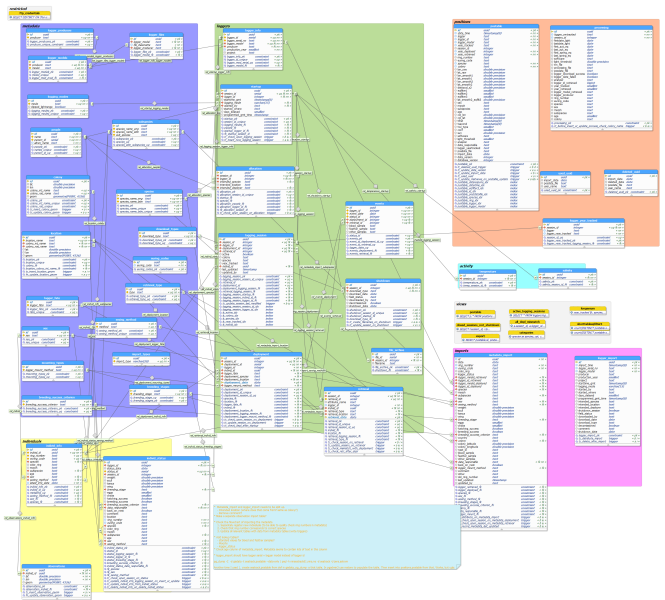
- Making your own custom queries of course requires some knowledge of how the database is structured.
- If you wan't to know more about this, pgAdmin can be a good tool to get an overview.
- We can also help you construct the queries that you are interested in, or give guidance.
- If a query is usted often, we can make a custom function in the R-package.
- The database structural model can be viewed by the command:

```
viewDatabaseModel()
```

Database structure

- The database structural model can be viewed by the command:

```
viewDatabaseModel()
```



Understanding this might still be challenging…

Database structure

- Most of the complexity deals with the logistical lifecycle of the loggers
  - This is handled in the schema “loggers”
  - Separate tables for startups, allocations, deployments, retrievals, shutdowns, associated filenames
- Much of the rest is lookup-tables, for data-integrity
  - Lookup tables in separate schema “metadata”
- Position data is in the “positions” schema, table “postable”
  - This contains all pre-processed position data.
- Info on individuals is in the “individuals” schema
  - Current info is stored in “individ_info”
  - Record of all status updates in “individ_status” (breeding, size, etc.)
- Most data can be linked (merged/joined) by the session_id
  - “Logger_id” and “individ_id” also useful
  - NB! that the position data is linked with the rest through the “session_id” column

NINA

Working with the database through the R-package

To simplify the usage of the database, we have created some R functions to read and write from the database. For example, to get a list all the active logger sessions (logger started up, but not yet shut down.):

```
activeSessions <- getActiveSessions()
activeSessions
# A tibble: 6,849 x 12
   id    session_id logger_id deployment_id retrieval_id active colony species
   <chr>      <int>     <int>         <int>        <int> <lgl>  <chr>  <chr>
 1 706f~     122133      3491        105367           NA TRUE   Hjelm~ Common~
 2 7070~     122134      3492        105368           NA TRUE   Hjelm~ Common~
 3 7072~     122135      3503        105369           NA TRUE   Hjelm~ Common~
 4 7073~     122136      3600        105370           NA TRUE   Hjelm~ Common~
 5 7075~     122137      3601        105371           NA TRUE   Hjelm~ Common~
 6 7076~     122138      3602        105372           NA TRUE   Hjelm~ Common~
 7 7078~     122139      3603        105373           NA TRUE   Hjelm~ Common~
 8 e0e9~     118929       966        102866           NA TRUE   Bear ~ Common~
 9 05ba~     119641      1583        103500           NA TRUE   Eynha~ Northe~
10 05bd~     119644      1586        103523           NA TRUE   Eynha~ Northe~
# ... with 6,839 more rows, and 4 more variables: year_tracked <chr>,
#   individ_id <chr>, last_updated <dttm>, updated_by <chr>
```


NINA

Working with the database through the R-package

Getting position data:

```
lbbg2015 <- getPosdata(
  selectSpecies = "Lesser black-backed gull",
  selectYear = "2015_16"
)
lbbg2015
# A tibble: 7,017 x 45
   id    date_time           logger logger_id logger_model year_tracked
   <chr> <dttm>              <chr>  <chr>     <chr>        <chr>
 1 f5c7~ 2016-03-13 22:05:48 S286_~ S286      c250         2015_16
 2 4131~ 2016-03-26 09:59:20 S286_~ S286      c250         2015_16
 3 d333~ 2016-05-12 09:26:29 S286_~ S286      c250         2015_16
 4 b5c7~ 2016-03-11 10:03:17 S286_~ S286      c250         2015_16
 5 ba14~ 2016-04-09 09:51:53 S286_~ S286      c250         2015_16
 6 6e0c~ 2016-04-08 21:48:01 S286_~ S286      c250         2015_16
 7 0929~ 2016-04-08 09:47:31 S286_~ S286      c250         2015_16
 8 7c3b~ 2016-04-07 21:56:22 S286_~ S286      c250         2015_16
 9 3df8~ 2016-04-07 09:47:44 S286_~ S286      c250         2015_16
10 0063~ 2016-04-06 21:46:44 S286_~ S286      c250         2015_16
# ... with 7,007 more rows, and 39 more variables: session_id <int>,
#   year_deployed <int>, year_retrieved <int>, ring_number <chr>,
#   euring_code <chr>, species <chr>, colony <chr>, lon_raw <dbl>,
#   lat_raw <dbl>, lon_smooth1 <dbl>, lat_smooth1 <dbl>, lon_smooth2 <dbl>,
#   lat_smooth2 <dbl>, disttocol_s2 <dbl>, eqfilter1 <int>, eqfilter2 <int>,
#   eqfilter3 <int>, lat_smooth2_eqfilt3 <dbl>, sex <chr>, morph <chr>,
#   subspecies <chr>, age <chr>, col_lon <dbl>, col_lat <dbl>, tfirst <dttm>,
#   tsecond <dttm>, twl_type <int>, conf <int>, sun <dbl>, software <chr>,
```

Working with the database through the R-package

Loading it with geometries as an sf object

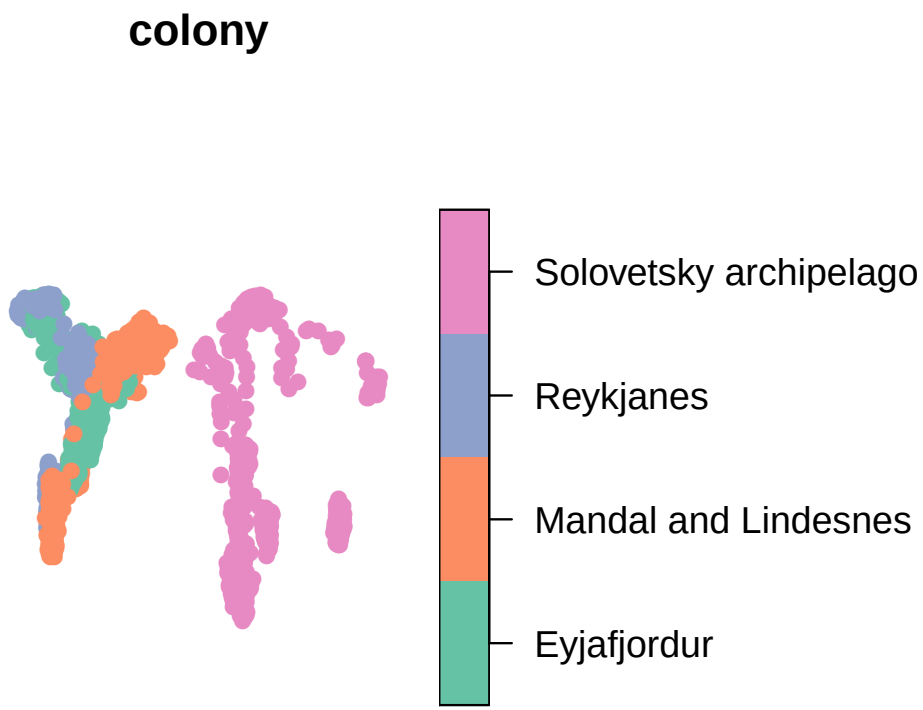
```
lbbg2015sf <- getPosdata(
  selectSpecies = "Lesser black-backed gull",
  selectYear = "2015_16",
  loadGeometries = T
)

lbbg2015sf
Simple feature collection with 4808 features and 45 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: -24.57907 ymin: -2.361598 xmax: 68.10425 ymax: 68.07847
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
# A tibble: 4,808 x 46
   id    date_time           logger logger_id logger_model year_tracked
   <chr> <dttm>              <chr>  <chr>     <chr>        <chr>
 1 d333~ 2016-05-12 09:26:29 S286_~ S286      c250         2015_16
 2 92cf~ 2016-01-20 09:32:03 S284_~ S284      c250         2015_16
 3 15ec~ 2016-01-19 21:32:33 S284_~ S284      c250         2015_16
 4 177f~ 2016-01-19 09:31:03 S284_~ S284      c250         2015_16
 5 d68e~ 2016-01-18 21:30:33 S284_~ S284      c250         2015_16
 6 c5c9~ 2016-01-18 09:29:33 S284_~ S284      c250         2015_16
 7 3218~ 2016-01-17 21:31:03 S284_~ S284      c250         2015_16
 8 ade1~ 2016-01-17 09:31:03 S284_~ S284      c250         2015_16
 9 cea8~ 2016-01-16 21:28:33 S284_~ S284      c250         2015_16
10 e485~ 2016-01-16 09:30:03 S284_~ S284      c250         2015_16
```

Working with the database through the R-package

- Plotting - native sf way

```
plot(lbbg2015sf["colony"],
  pch = 16,
  key.width = lcm(6)
)
```



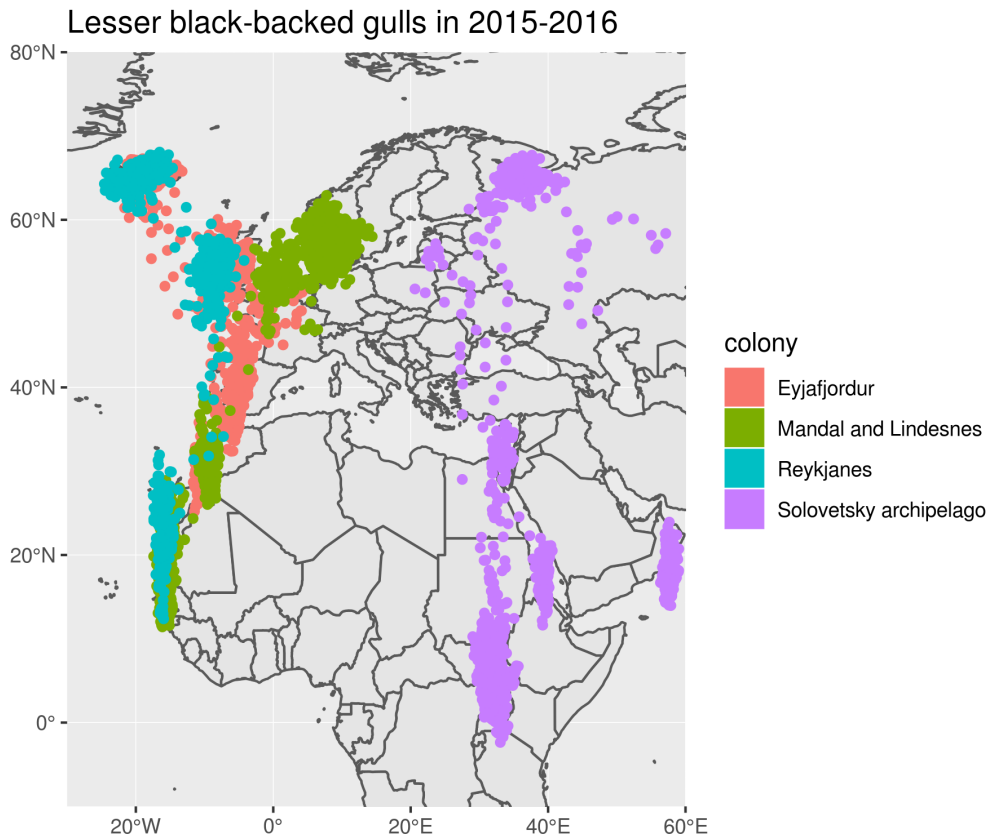
Working with the database through the R-package

- Plotting - ggplot2

```
require("rnaturalearth")
world <- ne_countries(scale = "medium", returnclass = "sf")
p <- ggplot(world) +
  geom_sf() +
  geom_sf(data = lbbg2015sf, aes(
    color = colony,
    fill = colony
  )) +
  coord_sf(xlim = c(-30, 60), ylim = c(-10, 80), expand = FALSE) +
  ggtitle("Lesser black-backed gulls in 2015-2016")
```


NINA

Working with the database through the R-package



Working with the database through the R-package

- Several other custom R functions exists, see help(package = “seatrackR”) for an overview. Look also at the vignettes
  - For example: `getIndividInfo`, `getLoggerInfo`
- Several functions for the few users that imports data

File archive

- In addition to the database, we also have an FTP-server (file archive) that can store the raw data files from the loggers
- After shutdown, each session is expected to yield a set of files, which is noted in the loggers.file_archive table
- The files should after that be given the correct names and be uploaded to the FTP-server
  - Custom function in the R-package: `uploadFiles`
- The FTP-server uses SSL security, and the R functions gets the login credentials from the PostgreSQL database
  - In other words, use the R functions to upload and download files from the file archive
  - No need for separate user credentials
  - Pretty good security

File archive

To see what the file archive contains (and not contains):

```
fileArchive <- listFileArchive()
```

```
fileArchive$filesInArchive
# A tibble: 19,796 x 1
   filename
   <chr>
 1 12141_2010_mk13.act
 2 12141_2010_mk13.lig
 3 12141_2010_mk13.trn
 4 12141_2010_mk13.txt
 5 12141_2011_mk13.act
 6 12141_2011_mk13.lig
 7 12141_2011_mk13.trn
 8 12141_2011_mk13.txt
 9 12143_2010_mk13.act
10 12143_2010_mk13.lig
# ... with 19,786 more rows
```

```
fileArchive$filesNotInArchive
# A tibble: 1,361 x 1
   filename
   <chr>
 1 B1142_2018_mk3006.txt
 2 B1152_2018_mk3006.txt
 3 B1158_2018_mk3006.txt
 4 B2467_2018_mk3006.txt
 5 B2475_2018_mk3006.txt
 6 B2479_2018_mk3006.txt
 7 C1159_2018_mk4083.txt
 8 C1165_2018_mk4083.txt
 9 C1807_2018_mk4083.txt
10 C1876_2018_mk4083.txt
# ... with 1,351 more rows
```

```
fileArchive$filesNotInDatabase
# A tibble: 2,757 x 1
   filename
   <chr>
 1 13022_2012_mk13.act
 2 13022_2012_mk13.lig
 3 13022_2012_mk13.trn
 4 13022_2012_mk13.txt
 5 13076_2010_mk13.TXT
 6 13076_2010_mk13.act
 7 13076_2010_mk13.lig
 8 13076_2010_mk13.trn
 9 13077_2010_mk13.TXT
10 13084_2010_mk13.TXT
# ... with 2,747 more rows
```

File archive

To get a summary of the expected files and their related info:

```
filesSummary <- getFileArchiveSummary()
filesSummary
```

```
# A tibble: 18,400 x 9
  file_id session_id colony      ring_number euring_code year_tracked
    <int>      <int> <chr>       <chr>       <chr>        <chr>
1  151880     117910 Alkefjellet 4182654     NOS          2015_16
2  151881     117910 Alkefjellet 4182654     NOS          2015_16
3  151882     117910 Alkefjellet 4182654     NOS          2015_16
4  151883     117910 Alkefjellet 4182654     NOS          2015_16
5  151884     117913 Alkefjellet 4182652     NOS          2015_16
  logger_serial_no logger_model filename
  <chr>            <chr>        <chr>
1 C101             mk4083       C101_2016_mk4083.lig
2 C101             mk4083       C101_2016_mk4083.act
3 C101             mk4083       C101_2016_mk4083.txt
4 C101             mk4083       C101_2016_mk4083.trn
5 C104             mk4083       C104_2016_mk4083.lig
# ... with 1.84e+04 more rows
```

File archive

Example: get the raw files from Røst in season 2014 - 2015.

First we check which files contains this information and see which ones exists in the file archive

```
rost2014ExpectedFiles <- filesSummary %>%
  filter(
    colony == "Rost",
    year_tracked == "2014_15"
  )
# merge with available files
rost2014AvailableFiles <- rost2014ExpectedFiles %>%
  inner_join(fileArchive$filesInArchive)
# all there?
nrow(rost2014ExpectedFiles)
[1] 144
nrow(rost2014AvailableFiles)
[1] 144
```

File archive

Downloading the files into a local folder.

```
downloadFiles(
  files = rost2014AvailableFiles$filename,
  destFolder = "rostRawFiles"
)
```

File archive

We can also load the contents of a file into R using the `loadFile` function. Here we look at the second file in the list from Røst in 2014.

```
M970_2015Trn <- loadFile(rost2014AvailableFiles$filename[2],
  col_names = F
)

M970_2015Trn
# A tibble: 423 x 3
   X1                  X2          X3
   <chr>               <chr>    <dbl>
 1 30/07/14 22:16:22 Sunset       3
 2 31/07/14 00:11:12 Sunrise      2
 3 07/08/14 22:10:15 Sunset       3
 4 08/08/14 00:06:43 Sunrise      3
 5 09/08/14 21:26:16 Sunset       9
 6 10/08/14 00:52:47 Sunrise      4
 7 11/08/14 21:29:17 Sunset       3
 8 12/08/14 01:43:29 Sunrise      3
 9 16/08/14 01:38:42 Sunrise      3
10 18/08/14 20:14:20 Sunset       3
# ... with 413 more rows
```

File archive

Note that some files have some initial information in a header and special format, that you have to specify.

```
M970_2015Sst <- loadFile(rost2014AvailableFiles$filename[1],
  col_names = F
)

M970_2015Sst %>% print(n = 12)
# A tibble: 889 x 1
   X1
   <chr>
 1 Migrate Technology Ltd logger
 2 Type: 4.44.8
 3 Logger number: M972
 4 MODE: 6 (clipped range light
 5 LIGHT: Sampled every minute with max light recorded every 5mins. Light readi~
 6 TEMPERATURE: Immersion max
 7 WET/DRY: Sampled every 30secs with number of samples wet recorded every 10mi~
 8 Max record length = 60 months. Total battery life upto 84 months. Logger is ~
 9 Programmed: 27/05/2014 23:58:36. Start of logging (DD/MM/YYYY HH:MM:SS): 27/~
10 Age at start of logging (secs): 259056
11 End of logging (DD/MM/YYYY HH:MM:SS): 06/07/2015 08:14:14
12 Age at end of logging (secs): 35194172
# ... with 877 more rows
```

File archive

Specifying rows to skip and custom column delimination.

```
M970_2015Sst <- loadFile(rost2014AvailableFiles$filename[1],
  col_names = T,
  skip = 19,
  delim = "\t"
)

M970_2015Sst
# A tibble: 871 x 5
   `DD/MM/YYYY HH:MM:S~ `wet min('C)` `wet max('C)` `wet mean('C)` `num samples`
   <chr>                        <dbl>         <dbl>          <dbl>         <dbl>
 1 19/06/2014 15:58:36           8             8.12           8.08             3
 2 20/06/2014 19:58:36           8.12          8.12           8.12             3
 3 20/06/2014 23:58:36           8.12          8.12           8.12             1
 4 21/06/2014 07:58:36           8.12          8.12           8.12             3
 5 21/06/2014 11:58:36           8.12          8.12           8.12             3
 6 21/06/2014 19:58:36           8.25          8.38           8.29             3
 7 22/06/2014 03:58:36           8.75          9              8.88             2
 8 22/06/2014 07:58:36           8.5           8.5            8.5              1
 9 22/06/2014 19:58:36           8.5           8.75           8.67             3
10 22/06/2014 23:58:36           8.62          8.62           8.62             1
# ... with 861 more rows
```

Foto på side 1: P. Jordhøy, J. Thomassen, E. B. Thorstad, A. Staverløkk, T. Aarvak, B. Løken/Samfoto