The MOISCRUST dataset: a spatio-temporal continuous soil moisture dataset from a Mediterranean semiarid dryland from 2006 to 2020 SUPPLEMENTARY MATERIAL

Joaquín Moreno[1,3] Sergio Asensio[3] Miguel Berdugo[3,4]
Beatriz Gozalo[3] Victoria Ochoa[3] Blas M. Benito[3,2]
Fernando T. Maestre[3,5]

Contents

[1] Corresponding author, e-mail: joaquin.moreno@ua.es
[2] Code and workflow design.
[3] Instituto Multidisciplinar para el Estudio del Medio "Ramon Margalef", Universidad de Alicante, Edificio Nuevos Institutos, Carretera de San Vicente del Raspeig s/n, 03690 San Vicente del Raspeig, Spain.
[4] Institut Department of Environmental Systems Science, ETH Zürich. Universitätstrasse 16, 8092 Zurich, Switzerland.
[5] Departamento de Ecología, Universidad de Alicante, Carretera de San Vicente del Raspeig s/n, 03690 San Vicente del Raspeig, Alicante, Spain.

1 Summary

The **MOISCRUST** database contains volumetric water content (VWC, m^3/m^3) measures taken by soil moisture sensors EC-5, Decagon Devices Inc., Pullman, USA) every 120 minutes (17th November 2006 to 31th January 2017) and 150 minutes (1st February 2017 to 16th December 2020) from three replicates in five different microsites (*Stipa* tussocks, *Retama* shrubs, and areas with low, medium and high cover of biocrust-forming lichens) located in The Aranjuez Experimental Station (central Iberian Peninsula, 40º02' N, 3º32'W; 590 m.a.s.l). During the long time-span these sensors have been working, there have been periods when data capture has not possible due to technical issues, and as consequence, 34.63% of the database records are missing entries. This reproducible workflow describes in detail the method used to impute missing data in the **MOISCRUST** database.

2 Reproducing this workflow

This reproducible workflow is available as an interactive Rstudio notebook in the file **moiscrust.Rmd** stored in this repository. It is packaged with renv to facilitate reproducibility. To run it in your computer, execute the code chunk below to prepare the session. You will need to replace reval = TRUEwithr eval = TRUE in the header of the code chunk.

```
install.packages("renv")
library(renv)
renv::restore()
```

3 Data loading and preparation

3.1 Installing and loading the required libraries

The following libraries are required to run this workflow. These are already installed in the local **renv** folder, so there is no need to install them if they are not available in your computer.

```
#automatic install of packages if they are not readily available
required_packages <- c(
 "data.table",
 "janitor",
 "tidyverse",
 "kableExtra",
 "foreach",
 "doParallel",
```

```
  "readr",
  "writexl",
  "RSQLite",
  "zip",
  "knitr",
  "DBI"
)

#loading packages
invisible(
 lapply(
   required_packages,
   library,
   character.only = TRUE
   )
 )

#knitr options
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE)
knitr::knit_hooks$set(document = function(x) {sub('\\usepackage{xcolor}', '\\usepackage{xcolor}', x, fixed = TRUE)})

#removing objects
rm(required_packages)
```

3.2 Loading and preparing the raw data

The raw data, stored in the file *moiscrust_raw.csv* was compiled by the members of Maestre Lab from the data provided by the soil moisture sensors.

```
#load, clean names, and rename time columns
moiscrust <- data.table::fread(file = "moiscrust_raw.csv") %>%
 janitor::clean_names() %>%
 dplyr::rename(
   date = dates,
   time = hour
   ) %>%
 as.data.frame()
```

date	time	retama5094	retama5062	retama5063	stipa5094	stipa5062	stipa5063	bs_cl5094	bs_cl5062	bs_cl5063	bs_cm5094	bs_cm5062	bs_cm5063	bs_ch5094	bs_ch5062	bs_ch5063
17/11/2006	18:00:00	0.197	NA	NA	0.132	NA	NA	0.232	NA	NA	0.205	NA	NA	0.121	NA	NA
17/11/2006	20:00:00	0.195	NA	NA	0.131	NA	NA	0.233	NA	NA	0.203	NA	NA	0.117	NA	NA
17/11/2006	22:00:00	0.194	NA	NA	0.131	NA	NA	0.234	NA	NA	0.203	NA	NA	0.117	NA	NA
18/11/2006	0:00:00	0.194	NA	NA	0.131	NA	NA	0.234	NA	NA	0.203	NA	NA	0.116	NA	NA
18/11/2006	2:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.203	NA	NA	0.116	NA	NA
18/11/2006	4:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
18/11/2006	6:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
18/11/2006	8:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
18/11/2006	10:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
18/11/2006	12:00:00	0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
18/11/2006	14:00:00	0.195	NA	NA	0.131	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
18/11/2006	16:00:00	0.195	NA	NA	0.131	NA	NA	0.233	NA	NA	0.201	NA	NA	0.115	NA	NA
18/11/2006	18:00:00	0.194	NA	NA	0.130	NA	NA	0.233	NA	NA	0.200	NA	NA	0.114	NA	NA
18/11/2006	20:00:00	0.193	NA	NA	0.129	NA	NA	0.233	NA	NA	0.199	NA	NA	0.112	NA	NA
18/11/2006	22:00:00	0.192	NA	NA	0.129	NA	NA	0.234	NA	NA	0.199	NA	NA	0.111	NA	NA
19/11/2006	0:00:00	0.192	NA	NA	0.128	NA	NA	0.234	NA	NA	0.199	NA	NA	0.111	NA	NA
19/11/2006	2:00:00	0.191	NA	NA	0.128	NA	NA	0.234	NA	NA	0.198	NA	NA	0.110	NA	NA
19/11/2006	4:00:00	0.190	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA
19/11/2006	6:00:00	0.189	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA
19/11/2006	8:00:00	0.188	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA

3.3 Formatting dates and times

The raw data contains two fields representing time, namely *date* (year, month, and day), and *time* (hour, minute, and seconds). Below we format these fields according to the POSIX standard, and add new fields adding criteria to subset the data by time:

- *date_time*: Date and time in POSIX format.
- *date_time_id*: Unique identificator for each *date_time*.
- *year*: Year number.
- *year_day*: Day of the year.
- *month*: Month number.
- *month_day*: Day of the month.
- *week*: Week of the year.
- *week_day*: Day of the week.

```
#date to Year-month-day
moiscrust$date <- format(
 as.POSIXct(
  strptime(
   moiscrust$date,
   "%d/%m/%Y",
   tz = ""
   )
  ),
 format = "%Y-%m-%d"
 )

#time to Hour-Minute
moiscrust$time <- format(
 as.POSIXct(
  strptime(
   moiscrust$time,
   "%H:%M:%S",
```

```
    tz = ""
    )
  ),
 format = "%H:%M"
 )

#joining date and time
moiscrust$date_time <- as.POSIXct(
 paste(
  moiscrust$date,
  moiscrust$time
  ),
 format="%Y-%m-%d %H:%M"
 )

#unique id for each observation
moiscrust$date_time_id <- 1:nrow(moiscrust)

#creating year, month, and day related columns
moiscrust$year <- lubridate::year(moiscrust$date)
moiscrust$year_day <- lubridate::yday(moiscrust$date)
moiscrust$month <- lubridate::month(moiscrust$date)
moiscrust$month_day <- lubridate::mday(moiscrust$date)
moiscrust$week <- lubridate::week(moiscrust$date)
moiscrust$week_day <- lubridate::wday(moiscrust$date)
```

3.4 Reordering time and data columns and arranging the data into long format

At this point, the *MOISCRUST* data has 15 columns representing soils moisture measures (microsites per three replicates per microsite), and 10 columns representing *time*. The data is also structured in what is known as a "wide format". Below we reorder these columns to facilitate arranging the data into a "long format".

```
#names of the sensors
sensors <- c(
 "retama5094",
 "retama5062",
 "retama5063",
 "stipa5094",
 "stipa5062",
```

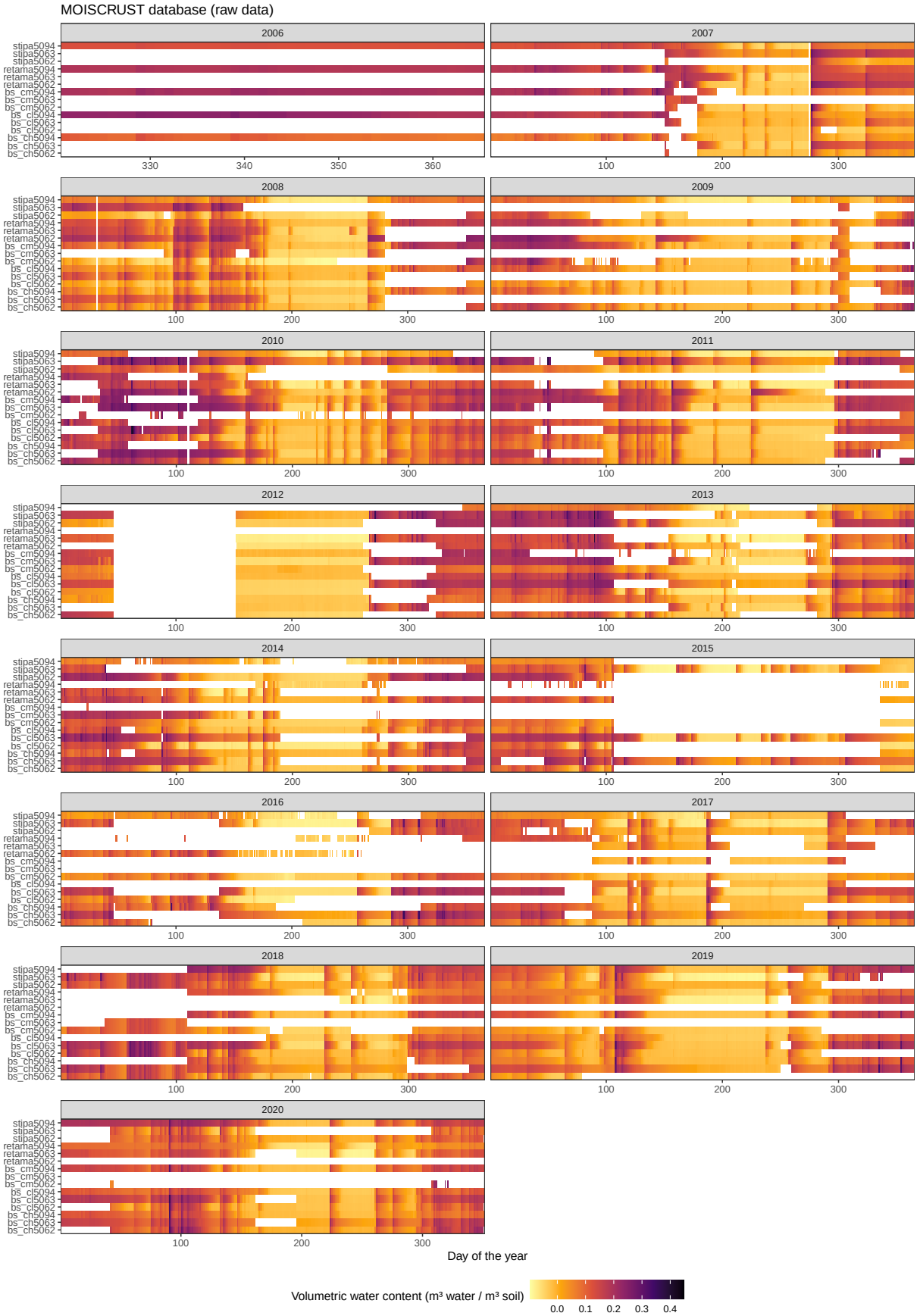
```
  "stipa5063",
  "bs_cl5094",
  "bs_cl5062",
  "bs_cl5063",
  "bs_cm5094",
  "bs_cm5062",
  "bs_cm5063",
  "bs_ch5094",
  "bs_ch5062",
  "bs_ch5063"
)

#reordering columns of moiscrust to have time in the left side
moiscrust <- moiscrust[, c(
  "date_time",
  "date_time_id",
  "date",
  "time",
  "year",
  "year_day",
  "month",
  "month_day",
  "week",
  "week_day",
  sensors
)]

#to long format
moiscrust_long <- tidyr::pivot_longer(
  moiscrust,
  cols = all_of(sensors),
  names_to = "sensor",
  values_to = "soil_moisture"
)
```

date_time	date_time_id	date	time	year	year_day	month	month_day	week	week_day	sensor	soil_moisture
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	retama5094	0.197
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	retama5062	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	retama5063	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	stipa5094	0.132
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	stipa5062	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	stipa5063	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cl5094	0.232
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cl5062	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cl5063	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cm5094	0.205
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cm5062	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_cm5063	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_ch5094	0.121
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_ch5062	NA
2006-11-17 18:00:00	1	2006-11-17	18:00	2006	321	11	17	46	6	bs_ch5063	NA
2006-11-17 20:00:00	2	2006-11-17	20:00	2006	321	11	17	46	6	retama5094	0.195
2006-11-17 20:00:00	2	2006-11-17	20:00	2006	321	11	17	46	6	retama5062	NA
2006-11-17 20:00:00	2	2006-11-17	20:00	2006	321	11	17	46	6	retama5063	NA
2006-11-17 20:00:00	2	2006-11-17	20:00	2006	321	11	17	46	6	stipa5094	0.131
2006-11-17 20:00:00	2	2006-11-17	20:00	2006	321	11	17	46	6	stipa5062	NA

3.5 Visualization of the raw data

Having the data in long format facilitates plotting and manipulation. The figure below shows the raw data, with missing data represented by the white color.

```
ggplot(moiscrust_long) +
  facet_wrap("year", scales = "free_x", ncol = 2) +
  aes(x = year_day, y = sensor, fill = soil_moisture) +
  geom_tile() +
  coord_cartesian(expand = FALSE) +
  theme_bw() +
  scale_fill_viridis_c(direction = -1, na.value = "white", option = "B") +
  theme(legend.position = "bottom") +
  ylab("") +
  xlab("Day of the year") +
  ggtitle("MOISCRUST database (raw data)") +
  labs(fill = expression("Volumetric water content (m³ water / m³ soil)")) +
  theme(legend.key.width = unit(1, "cm"))
```


MOISCRUST database (raw data)
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
stipa5094
stipa5063
stipa5062
retama5094
retama5063
retama5062
bs_cm5094
bs_cm5063
bs_cm5062
bs_cl5094
bs_cl5063
bs_cl5062
bs_ch5094
bs_ch5063
bs_ch5062
Day of the year
Volumetric water content (m³ water / m³ soil)
0.0
0.1
0.2
0.3
0.4

3.6 Number of NA per sensor

Due to technical constraints, there is a large number of missing data in the *MOISCRUST* dataset. To better understand the extent of such missing data, the code below counts the number of missing entries per sensor.

```
#counting NA values per sensor
moiscrust_NA <- moiscrust_long %>%
 group_by(sensor) %>%
 summarise(na_count = sum(is.na(soil_moisture))) %>%
 mutate(na_count_percent = round((na_count * 100) / nrow(moiscrust), 1))

#adding sensor microsite to the moiscrust_NA data frame
moiscrust_NA$microsite <- c(
 "biocrust_high",
 "biocrust_high",
 "biocrust_high",
 "biocrust_low",
 "biocrust_low",
 "biocrust_low",
 "biocrust_medium",
 "biocrust_medium",
 "biocrust_medium",
 "retama",
 "retama",
 "retama",
 "stipa",
 "stipa",
 "stipa"
)

#reordering columns and arranging by na_count
moiscrust_NA <- moiscrust_NA[, c(
 "sensor",
 "microsite",
 "na_count",
 "na_count_percent"
 )] %>%
 dplyr::arrange(na_count) %>%
 as.data.frame()
```

sensor	microsite	na_count	na_count_percent
bs_ch5094	biocrust_high	7674	16.5
stipa5094	stipa	10553	22.7
bs_cl5094	biocrust_low	10987	23.6
bs_cl5063	biocrust_low	11114	23.9
bs_cl5062	biocrust_low	11188	24.1
bs_ch5062	biocrust_high	11776	25.3
bs_ch5063	biocrust_high	14125	30.4
stipa5062	stipa	15363	33.0
stipa5063	stipa	15604	33.5
bs_cm5094	biocrust_medium	16934	36.4
bs_cm5062	biocrust_medium	18743	40.3
retama5063	retama	20505	44.1
retama5094	retama	22840	49.1
retama5062	retama	22999	49.4
bs_cm5063	biocrust_medium	31225	67.1

4 Imputation of missing data

The imputation of missing data we implement here to impute missing data in *MOISCRUST* works by finding, for a given entry *y* with missing data at a time *t*, the sensor *x* with data for *t* that is in the same type of microsite (if possible), has the longest extent in common, and the highest correlation with the sensor to which *y* belongs, and estimates *y* with the linear model *y ~ x*.

4.1 Developing criteria to find candidates for gap filling

To generate the criteria to find the best possible candidate *x* to estimate the missing data *y*, we compute the common length and correlation between all pairs of sensors, and generate a column indicating whether they belong to the same microsite or not. With the values stored in these columns we compute a *selection score* based on the following expression:

$$S_x = \%vc_{x,y} + (R^2_{x,y} * 100) + \begin{cases} 100, & \text{if } microsite_x == microsite_y \\ 0, & \text{otherwise} \end{cases}$$

Where:

- y is the sensor with a missing value to be estimated.
- x is the sensor to be used as candidate predictor to estimate the missing value in y.
- S_x is the selection score of the candidate sensor x, in the range [0, 300].
- $\%vc_{x,y}$ is the percent of common valid cases of the sensors x and y.

- $R^2_{x,y}$ is the Pearson's R² of the common valid cases of the sensors x and y.
- $microsite_x$ and $microsite_y$ are the respective microsites of the sensors x and y.

During data imputation, for each missing value, the sensor with the higher selection score is used to estimate it.

These criteria are stored in the data frame **sensors_pairs**. Along with the computation of the selection score, the code below also computes a linear model for each pair x y, and stores it in the object **sensors_pairs_models**. The identificators of these models are stored in the column *model_id* of the data frame **sensors_pairs**.

```
#combining sensors in pairs x-y
sensors_pairs <- combn(
  x = sensors,
  m = 2
) %>%
  t() %>%
  as.data.frame()

#adding combinations y-x so all pairs have both directions
#removing repeated pairs
#joining with moiscrust_NA to get sensor groups
#add column same_microsite to check if x and y are or not in the same sensor group
#add empty columns to store % of shared data, model's R squared, and model ID
sensors_pairs <- sensors_pairs %>%
  rbind(
    data.frame(
      V1 = sensors_pairs$V2,
      V2 = sensors_pairs$V1
    )
  ) %>%
  distinct(
    V1,
    V2,
    .keep_all = TRUE
  ) %>%
  left_join(
    moiscrust_NA[, c("sensor", "microsite")],
    by = c("V1" = "sensor")
  ) %>%
  left_join(
    moiscrust_NA[, c("sensor", "microsite")],
    by = c("V2" = "sensor")
```

```
 ) %>%
 rename(
  y = V1,
  x = V2,
  y_microsite = microsite.x, #not a mistake
  x_microsite = microsite.y, #not a mistake
 ) %>%
 mutate(
  same_microsite = ifelse(
    x_microsite == y_microsite,
    TRUE,
    FALSE
    ),
  sensors_shared_valid_percent = NA,
  sensors_r_squared = NA,
  model_id = row_number()
 )

#list to store models
sensors_pairs_models <- list()

#looping through sensors pairs to:
#fit lm model y ~ x and save it in sensors_pairs_models
#
for(i in 1:nrow(sensors_pairs)){

 #names of the sensors y and x
 y_i <- sensors_pairs[i, "y"]
 x_i <- sensors_pairs[i, "x"]

 #data of the sensor pair
 sensor_pair_i <- moiscrust[, c(y_i, x_i)]

 #complete cases of the sensor pair
 sensor_pair_i <- sensor_pair_i[complete.cases(sensor_pair_i), ]

 #common cases
 sensors_pairs[i, "sensors_shared_valid_percent"] <-
  nrow(sensor_pair_i) / nrow(moiscrust) * 100

 #R squared of the sensor pair
```

```
  sensors_pairs[i, "sensors_r_squared"] <- cor(
    sensor_pair_i[, 1],
    sensor_pair_i[, 2]
    )

  #model formula y ~ x
  formula_i <- as.formula(paste(y_i, "~", x_i))

  #linear model
  sensors_pairs_models[[i]] <- lm(
    formula = formula_i,
    data = sensor_pair_i
  )

}

#selection score to find candidates during gap filling
#(sensors_r_squared * 100) +
#sensors_shared_valid_percent +
#same_microsite (TRUE = 100, FALSE = 0)
sensors_pairs <- mutate(
  sensors_pairs,
  selection_score =
    (sensors_r_squared * 100) +
    sensors_shared_valid_percent +
    ifelse(same_microsite == TRUE, 100, 0)
)

#removing objects we don't need
rm(
  sensor_pair_i,
  formula_i,
  i,
  x_i,
  y_i
)
```

The resulting data frame, named **sensors_pairs**, has columns with the names of the sensor *y* (the one with missing data to impute), the sensor *x* (the candidate to be used as predictor to estimate *y*), their respective microsites, a column indicating if they belong to the same microsite, the percent of shared valid data, the R squared of their shared data, a

y	x	y_microsite	x_microsite	same_microsite	sensors_shared_valid_percent	sensors_r_squared	model_id	selection_score
retama5094	retama5062	retama	retama	TRUE	20.503945	0.8825786	1	208.76181
retama5094	retama5063	retama	retama	TRUE	28.736052	0.9065618	2	219.39223
retama5094	stipa5094	retama	stipa	FALSE	49.437792	0.6546707	3	114.90486
retama5094	stipa5062	retama	stipa	FALSE	34.751575	0.6570769	4	100.45926
retama5094	stipa5063	retama	stipa	FALSE	27.985724	0.8494829	5	112.93401
retama5094	bs_cl5094	retama	biocrust_low	FALSE	45.825898	0.8610431	6	131.93021
retama5094	bs_cl5062	retama	biocrust_low	FALSE	39.137445	0.6276224	7	101.89968
retama5094	bs_cl5063	retama	biocrust_low	FALSE	34.364586	0.8118848	8	115.55307
retama5094	bs_cm5094	retama	biocrust_medium	FALSE	44.099499	0.8808309	9	132.18258
retama5094	bs_cm5062	retama	biocrust_medium	FALSE	31.105282	0.7429876	10	105.40404
retama5094	bs_cm5063	retama	biocrust_medium	FALSE	5.892976	0.8384753	11	89.74050
retama5094	bs_ch5094	retama	biocrust_high	FALSE	48.414422	0.7630192	12	124.71634
retama5094	bs_ch5062	retama	biocrust_high	FALSE	38.305420	0.7500951	13	113.31493
retama5094	bs_ch5063	retama	biocrust_high	FALSE	32.943478	0.8490180	14	117.84528
retama5062	retama5063	retama	retama	TRUE	32.057704	0.8394976	15	216.00746
retama5062	stipa5094	retama	stipa	FALSE	42.633242	0.7653060	16	119.16384
retama5062	stipa5062	retama	stipa	FALSE	37.522843	0.5339814	17	90.92099
retama5062	stipa5063	retama	stipa	FALSE	31.636317	0.8561540	18	117.25171
retama5062	bs_cl5094	retama	biocrust_low	FALSE	43.684561	0.7455036	19	118.23492
retama5062	bs_cl5062	retama	biocrust_low	FALSE	48.896008	0.6940089	20	118.29689

4.2 Generating the x and y matrices to predict imputed values

During data imputation, two data frames are needed. The data frame **x** contains the data of every sensor for every available time, while the data frame **y**, which starts with empty values, is where the imputed values, their confidence intervals, selection criteria, and other quality-related columns are going to be stored.

```
#creating data frame of predictors
x <- moiscrust[, sensors]

#creating data frame to store model results
y <- matrix(
 data = NA,
 nrow = nrow(moiscrust),
 ncol = 12
 ) %>%
 as.data.frame()

#new colnames
colnames(y) <- c(
 "interpolated",
 "model_estimate",
 "model_ci_lower",
 "model_ci_upper",
 "model_predictor",
 "same_microsite",
 "sensors_r_squared",
 "sensors_shared_valid_percent",
 "selection_score",
```

```
 "date_time_id",
 "sensor",
 "microsite"
 )

#transferring time id
y[, "date_time_id"] <- moiscrust[, "date_time_id"]
y[, "interpolated"] <- FALSE
```

The **x** data frame looks as follows:

retama5094	retama5062	retama5063	stipa5094	stipa5062	stipa5063	bs_cl5094	bs_cl5062	bs_cl5063	bs_cm5094	bs_cm5062	bs_cm5063	bs_ch5094	bs_ch5062	bs_ch5063
0.197	NA	NA	0.132	NA	NA	0.232	NA	NA	0.205	NA	NA	0.121	NA	NA
0.195	NA	NA	0.131	NA	NA	0.233	NA	NA	0.203	NA	NA	0.117	NA	NA
0.194	NA	NA	0.131	NA	NA	0.234	NA	NA	0.203	NA	NA	0.117	NA	NA
0.194	NA	NA	0.131	NA	NA	0.234	NA	NA	0.203	NA	NA	0.116	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.203	NA	NA	0.116	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.115	NA	NA
0.194	NA	NA	0.130	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
0.195	NA	NA	0.131	NA	NA	0.234	NA	NA	0.202	NA	NA	0.116	NA	NA
0.195	NA	NA	0.131	NA	NA	0.233	NA	NA	0.201	NA	NA	0.115	NA	NA
0.194	NA	NA	0.130	NA	NA	0.233	NA	NA	0.200	NA	NA	0.114	NA	NA
0.193	NA	NA	0.129	NA	NA	0.233	NA	NA	0.199	NA	NA	0.112	NA	NA
0.192	NA	NA	0.129	NA	NA	0.234	NA	NA	0.199	NA	NA	0.111	NA	NA
0.192	NA	NA	0.128	NA	NA	0.234	NA	NA	0.199	NA	NA	0.111	NA	NA
0.191	NA	NA	0.128	NA	NA	0.234	NA	NA	0.198	NA	NA	0.110	NA	NA
0.190	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA
0.189	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA
0.188	NA	NA	0.127	NA	NA	0.234	NA	NA	0.198	NA	NA	0.109	NA	NA

And this is the **y** data frame, that will be filled during data imputation:

interpolated	model_estimate	model_ci_lower	model_ci_upper	model_predictor	same_microsite	sensors_r_squared	sensors_shared_valid_percent	selection_score	date_time_id	sensor	microsite
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	1	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	2	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	3	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	4	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	5	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	6	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	7	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	8	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	9	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	10	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	11	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	12	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	13	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	14	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	15	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	16	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	17	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	18	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	19	NA	NA
FALSE	NA	NA	NA	NA	NA	NA	NA	NA	20	NA	NA

4.3 Data imputation, step by step

The steps to fill the gaps in the **MOISCRUST** database go as follows:

1. A given sensor name is selected: "stipa5063"

```
sensor = "stipa5063"
```

2. The sensors pairs from the table **sensors_pairs** where the selected sensor is *y* (the sensor which values are to be imputed) are selected.

```
sensors_pair <- sensors_pairs %>%
  dplyr::filter(y == sensor)
```

y	x	y_microsite	x_microsite	same_microsite	sensors_shared_valid_percent	sensors_r_squared	model_id	selection_score
stipa5063	bs_cl5094	stipa	biocrust_low	FALSE	47.42330	0.7826120	61	125.6845
stipa5063	bs_cl5062	stipa	biocrust_low	FALSE	52.09941	0.6922529	62	121.3247
stipa5063	bs_cl5063	stipa	biocrust_low	FALSE	66.13635	0.8021262	63	146.3490
stipa5063	bs_cm5094	stipa	biocrust_medium	FALSE	41.85712	0.8607496	64	127.9321
stipa5063	bs_cm5062	stipa	biocrust_medium	FALSE	42.63754	0.7499577	65	117.6333
stipa5063	bs_cm5063	stipa	biocrust_medium	FALSE	26.90646	0.8956584	66	116.4723
stipa5063	bs_ch5094	stipa	biocrust_high	FALSE	52.89274	0.7012173	67	123.0145
stipa5063	bs_ch5062	stipa	biocrust_high	FALSE	51.91452	0.7804496	68	129.9595
stipa5063	bs_ch5063	stipa	biocrust_high	FALSE	59.66504	0.7433338	69	133.9984
stipa5063	retama5094	stipa	retama	FALSE	27.98572	0.8494829	110	112.9340
stipa5063	retama5062	stipa	retama	FALSE	31.63632	0.8561540	123	117.2517
stipa5063	retama5063	stipa	retama	FALSE	45.96779	0.8168872	135	127.6565
stipa5063	stipa5094	stipa	stipa	TRUE	48.28543	0.5162195	146	199.9074
stipa5063	stipa5062	stipa	stipa	TRUE	48.22953	0.5211191	156	200.3414

3. The first row of the data frame **x** is selected.

```
x_row <- x[1, ]
```

retama5094	retama5062	retama5063	stipa5094	stipa5062	stipa5063	bs_cl5094	bs_cl5062	bs_cl5063	bs_cm5094	bs_cm5062	bs_cm5063	bs_ch5094	bs_ch5062	bs_ch5063
0.197	NA	NA	0.132	NA	NA	0.232	NA	NA	0.205	NA	NA	0.121	NA	NA

3.1. If there is a valid value of soil moisture for the sensor “stipa5063”, the algorithm goes to the next row, until there is a row with a missing value.

4. If there is an missing value (*NA*), the potential candidate predictors are selected from the row by removing the data of other sensors with *NA*, and the data of the target sensor.

```
predictor_candidates <- as.vector(x[1, ])
predictor_candidates <- predictor_candidates[which(
    !is.na(predictor_candidates) &
      names(predictor_candidates) != sensor
    )]
```

retama5094	stipa5094	bs_cl5094	bs_cm5094	bs_ch5094
0.197	0.132	0.232	0.205	0.121

5. From these predictors, the one with the highest selection score is selected from the data frame **sensors_pair** generated in the step **2.**.

```
best_predictor <- sensors_pair %>%
  dplyr::filter(x %in% names(predictor_candidates)) %>%
  dplyr::arrange(desc(selection_score)) %>%
  dplyr::slice(1)
```

y	x	y_microsite	x_microsite	same_microsite	sensors_shared_valid_percent	sensors_r_squared	model_id	selection_score
stipa5063	stipa5094	stipa	stipa	TRUE	48.28543	0.5162195	146	199.9074

6. The model to use, stored in the list **sensors_pairs_model**, is selected from *model_id* column of the **best_predictor** data frame , and used to predict a value for the empty cell.

```
predict(
  object = sensors_pairs_models[[best_predictor$model_id]],
  newdata = x_row,
  se.fit = TRUE,
  type = "response",
  interval = "confidence"
  )$fit
```

```
##        fit      lwr      upr
## 1 0.1435911 0.1419916 0.1451907
```

7. The imputed value, its confidence intervals, and other values about the imputation quality available in **best_predictor** are transferred to the same row in the data frame **y**.

8. Once all the sensors and rows have been processed this way, the matrix **y** is joined with **moiscrust_long**, and its interpolated values are transferred to the *soil_moisture* column, along with other columns indicating the quality of the interpolation.

4.4 Applying the algorithm to the complete dataset

The code below applies the algorithm to every sensor and row with missing data. Sensors are processed in parallel to speed up the data imputation operation.

```
#setup for parallel execution
if(.Platform$OS.type == "windows"){
  temp_cluster <- parallel::makeCluster(
    parallel::detectCores() - 1,
    type = "PSOCK"
  )
} else {
  temp_cluster <- parallel::makeCluster(
    parallel::detectCores() - 1,
    type = "FORK"
  )
}
doParallel::registerDoParallel(cl = temp_cluster)

#parallelized loop (each sensor is processed in one separated thread)
moiscrust_interpolation <- foreach::foreach(
  sensor_i = sensors,
  .packages = "tidyverse"
) %dopar% {
```

```
#subset sensors_pairs
sensors_pair_i <- sensors_pairs %>%
 dplyr::filter(y == sensor_i)

#fill microsite
y[, "microsite"] <-sensors_pair_i$y_microsite[1]

#scanning the rows of x one by one
for(row_i in 1:nrow(x)){

 #if is not NA, next iteration
 if(!is.na(x[row_i, sensor_i])){next}

 #getting target row row
 x_row_i <- x[row_i, ]

 #getting predictor candidates available in x_row_i
 predictor_candidates_i <- as.vector(x_row_i)
 predictor_candidates_i <- predictor_candidates_i[which(
  !is.na(predictor_candidates_i) &
   names(predictor_candidates_i) != sensor_i
 )]

 #selecting the predictor candidate with the best selection_score score
 best_predictor_i <- sensors_pair_i %>%
  dplyr::filter(x %in% names(predictor_candidates_i)) %>%
  dplyr::arrange(desc(selection_score)) %>%
  dplyr::slice(1)

 #if there is no best candidate available, next iteration
 if(nrow(best_predictor_i) == 0){next}

 #compute estimates with the model of the best predictor
 y[row_i, c(
  "model_estimate",
  "model_ci_lower",
  "model_ci_upper"
 )] <- predict(
   object = sensors_pairs_models[[best_predictor_i$model_id]],
   newdata = x_row_i,
   se.fit = TRUE,
```

```
      type = "response",
      interval = "confidence"
      )$fit

   #adding interpolation flag
   y[row_i, "interpolated"] <- TRUE
   y[row_i, "model_predictor"] <- best_predictor_i$x
   y[row_i, "sensors_r_squared"] <- best_predictor_i$sensors_r_squared
   y[row_i, "selection_score"] <- best_predictor_i$selection_score
   y[row_i, "sensors_shared_valid_percent"] <- best_predictor_i$sensors_shared_valid_percent
   y[row_i, "same_microsite"] <- best_predictor_i$same_microsite

  }

  #adding sensor_i name
  y[, "sensor"] <- sensor_i

  return(y)

}

#stop cluster
parallel::stopCluster(temp_cluster)

#removing loop objects
rm(
  x,
  y,
  temp_cluster
)
```

The imputation algorithm produces an object (a list) named **moiscrust_interpolation**, with one slot per sensor, each one with one **y** data frame containing the imputation results. Below we transform this object into the data frame **moiscrust_interpolation_long** and join it with **moiscrust_long**, to start preparing the database format.

```
#naming the output
names(moiscrust_interpolation) <- sensors

#to data frame
moiscrust_interpolation_long <- do.call(
  "rbind",
```

```
  moiscrust_interpolation
)

#joining with moiscrust_long
moiscrust_long <- dplyr::left_join(
  moiscrust_long,
  moiscrust_interpolation_long,
  by = c("date_time_id", "sensor")
)

#transferring estimates to the soil_moisture column
moiscrust_long$soil_moisture <- ifelse(
  is.na(moiscrust_long$soil_moisture),
  moiscrust_long$model_estimate,
  moiscrust_long$soil_moisture
)

#adding a interpolation_quality flag following the criteria in the paper
moiscrust_long$interpolation_quality <- ifelse(
  moiscrust_long$sensors_r_squared > 0.85 &
  moiscrust_long$sensors_shared_valid_percent > 20,
  "acceptable",
  "poor"
)

#filling NA with "observation"
moiscrust_long[
  is.na(moiscrust_long$interpolation_quality), "interpolation_quality"
  ] <- "observation"

#adding NA where there are no values
moiscrust_long[is.na(moiscrust_long$soil_moisture), "interpolation_quality"] <- NA

#computing number of NA cases again
moiscrust_NA <- moiscrust_long %>%
  group_by(sensor) %>%
  summarise(na_count_after = sum(is.na(soil_moisture))) %>%
  mutate(na_count_percent_after = round((na_count_after * 100) / nrow(moiscrust), 1)) %>%
  left_join(
    y = moiscrust_NA,
    by = "sensor"
```

```
) %>%
 transmute(
  sensor,
  na_count_before = na_count,
  na_count_after,
  na_count_percent_before = na_count_percent,
  na_count_percent_after
 )

#removing moiscrust_interpolation
rm(moiscrust_interpolation)
```

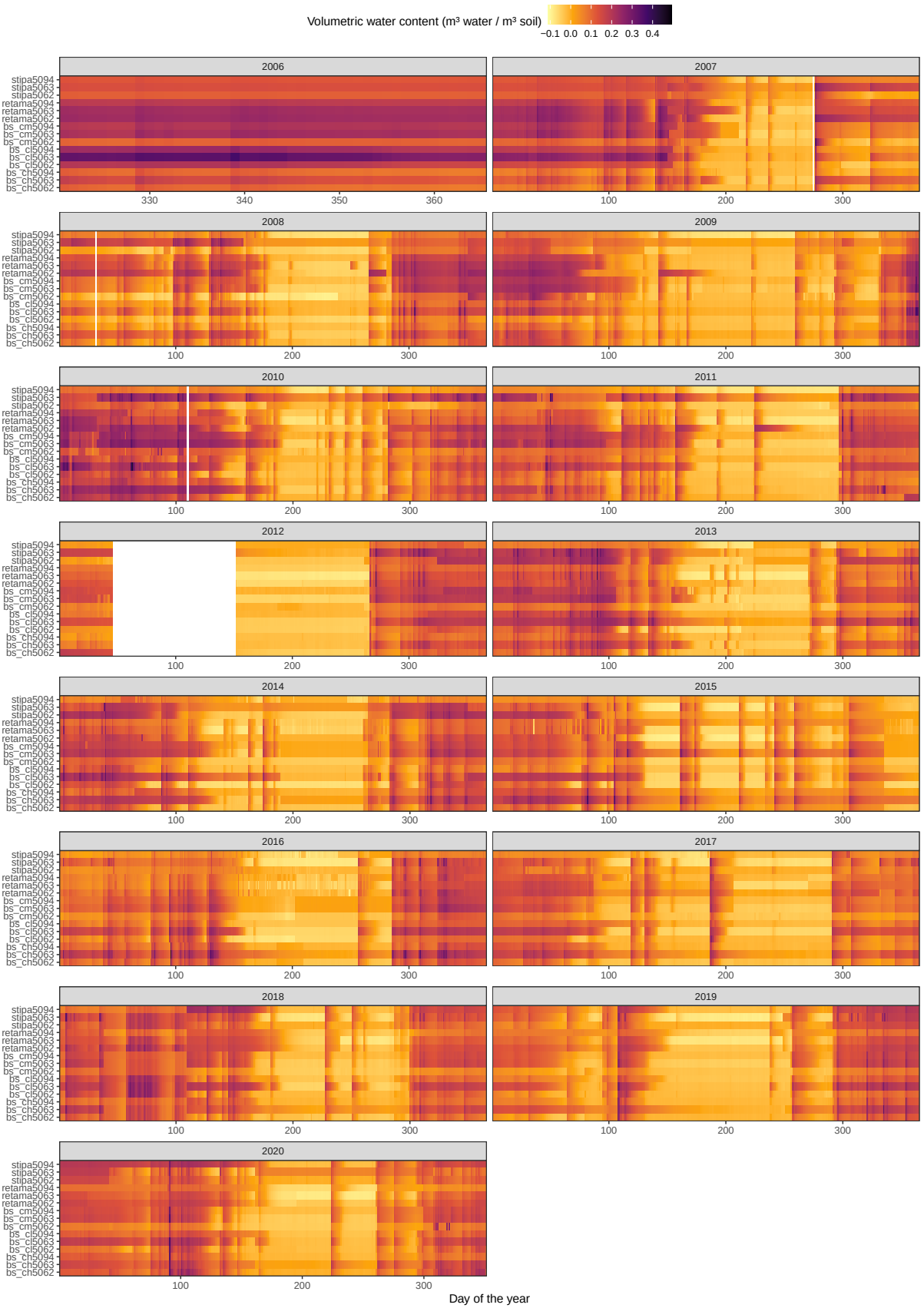
The interpolation has removed all gaps where there was a value to interpolate from, as shown in the table below.

Sensor	NA before interpolation	NA after interpolation	NA % before interpolation	NA % after interpolation
bs_ch5062	11776	989	25.3	2.1
bs_ch5063	14125	989	30.4	2.1
bs_ch5094	7674	989	16.5	2.1
bs_cl5062	11188	989	24.1	2.1
bs_cl5063	11114	989	23.9	2.1
bs_cl5094	10987	989	23.6	2.1
bs_cm5062	18743	989	40.3	2.1
bs_cm5063	31225	989	67.1	2.1
bs_cm5094	16934	989	36.4	2.1
retama5062	22999	989	49.4	2.1
retama5063	20505	989	44.1	2.1
retama5094	22840	989	49.1	2.1
stipa5062	15363	989	33.0	2.1
stipa5063	15604	989	33.5	2.1
stipa5094	10553	989	22.7	2.1

4.5 Visualizing the interpolated time series

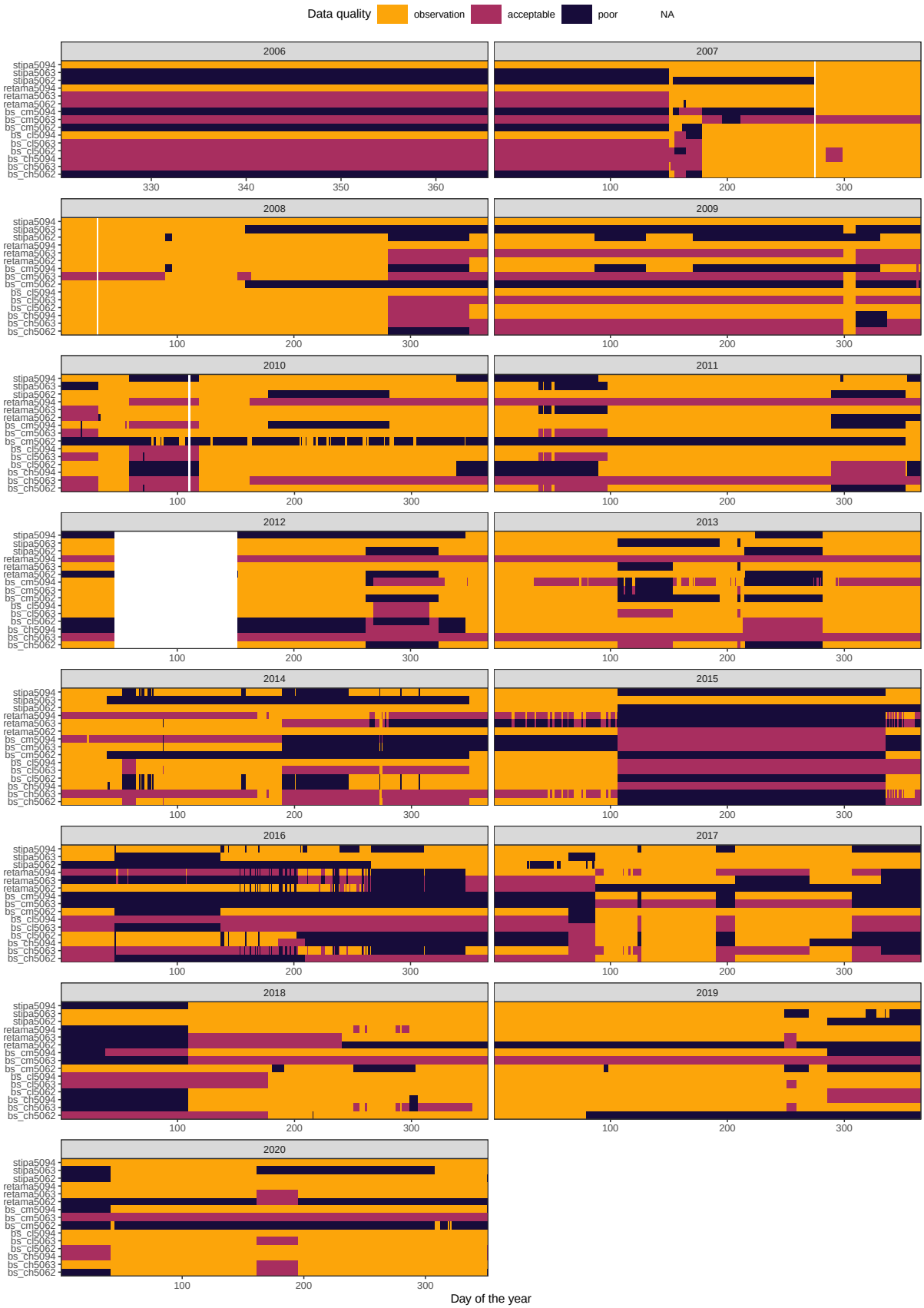
The **MOISCRUST** database looks as follows after applying the imputation algorithm.

MOISCRUST database (observed and interpolated records)



The plot above represents both observed and interpolated values, without a clear differentiation between each type. To provide an indicator of imputation quality, the imputation algorithm also generated a new column named *interpolation_quality*, where the observations are marked with the flag “observation”, the imputed data where *x* and *y* shared more than 20% of valid cases and had an R^2 higher than 0.85 are marked with the flag “acceptable”, and the imputed data below these thresholds are marked with the flag “poor”. The plot below shows the values of these flags for each sensor and time, year per year.

MOISCRUST database (data quality)



5 Data base format

The dataset **moiscrust_long** is the database in long format already. Now we can rename it to **moiscrust**.

```
moiscrust <- moiscrust_long
```

The database columns are:

- *date_time*: date and time in POSIX format.
- *date_time_id*: integer, unique ID for each value of *date_time*.
- *date*: date in format year-month-day.
- *time*: time in format hour-minute.
- *year*: integer, year.
- *year_day*: integer, day of the year.
- *month*: integer, month number.
- *week*: integer, week of the year.
- *week_day*: integer, day of the week.
- *sensor*: character, sensor name.
- *soil_moisture*: numeric, soil moisture value in $m^3 water / m^3 soil$.
- *interpolated*: boolean, *TRUE* for interpolated records and *FALSE* for observations.
- *model_estimate*: numeric, prediction of the linear model.
- *model_ci_lower*: numeric, lower bound of the confidence interval of the estimate.
- *model_ci_upper*: numeric, upper bound of the confidence interval of the estimate.
- *model_predictor*: character, name of the sensor used as predictor in the linear model.
- *same_microsite*: boolean, *TRUE* if the sensor and its predictor are in the same group ("stipa", "retama", "biocrust_low", "biocrust_medium", "biocrust_high").
- *sensors_r_squared*: numeric, R squared between *sensor* and *model_predictor*.
- *sensors_shared_valid_percent*: numeric, percentage of shared valid cases between *sensor* and *model_predictor*, taking the total number of values in *date_time_id* as reference.
- *selection_score*: numeric, value used to select the *model_predictor*, based on the sum of *same_microsite* (100 if *TRUE* and 0 if *FALSE*), *sensors_r_squared* multiplied by 100, and *sensors_shared_valid_percent*.
- *interpolation_quality*: character, with the values "observation" for observations, "acceptable" for interpolated values where *sensors_r_squared* is higher than 0.85 and *sensors_shared_valid_percent* is higher than 20, and "poor" for interpolated values below at least one of these thresholds.

5.1 Saving the data base in different formats

To expand its usability as much as possible, we provide the data in four different formats: .RData, .csv, .xlsx, and SQLite.

```r
#if the zip file does not exists, creates database directory and populates it
if(!file.exists("database.zip")){

 dir.create("database")

 #save as RData
 save(
  moiscrust,
  file = "database/moiscrust.RData"
 )

 #save as csv
 readr::write_excel_csv(
  x = moiscrust,
  file = "database/moiscrust.csv"
 )

 #save as excel file
 writexl::write_xlsx(
  x = moiscrust,
  path = "database/moiscrust.xlsx"
 )

 #save as SQLite
 db.driver <- DBI::dbDriver("SQLite")
 db.connection <-  DBI::dbConnect(
  db.driver,
  dbname = "database/moiscrust.db"
  )
 DBI::dbWriteTable(
  db.connection,
  "moiscrust",
  moiscrust,
  overwrite = TRUE
  )
 DBI::dbDisconnect(db.connection)

 #compressing the file
 zip::zip(
  zipfile = "database.zip",
  files = "database"
```

```
 )

 #removing the database folder
 unlink(
  "database",
  recursive = TRUE
  )

 #clean the R environment
 rm(
  db.connection,
  db.driver,
  moiscrust
  )

}
```

6 Usage examples

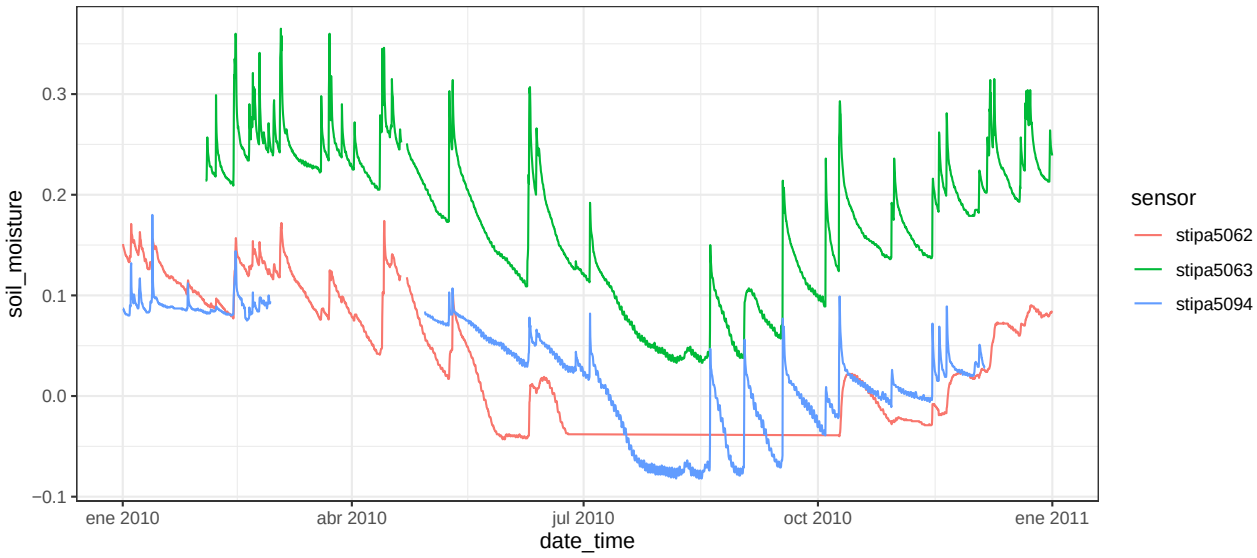
This section shows several simple ways to work with the database in R, focusing on the minimal code required to perform basic operations. But first we have to unzip **database.zip** and read **moiscrust.RData** into the R environment.

```
#unzip the zip file with the database files
zip::unzip("database.zip")

#load the moiscrust dataframe
load("database/moiscrust.RData")

#delete the database folder
 unlink(
  "database",
  recursive = TRUE
  )
```

6.1 Plotting time-series of observed data for a given microsite and year

The column *interpolated* contains TRUE for imputed data and FALSE for observed data, and therefore, it can be used to separate observed and imputed data right away. The column microsite groups together time-series belonging to the same microsite. Finally, the column sensor contains the data yielded by specific sensors.

```
moiscrust %>%
 dplyr::filter(
  interpolated == FALSE,
  microsite == "stipa",
  year == 2010
 ) %>%
 ggplot2::ggplot() +
 ggplot2::aes(
  x = date_time,
  y = soil_moisture,
  color = sensor
 ) +
 ggplot2::geom_line() +
 ggplot2::theme_bw()
```



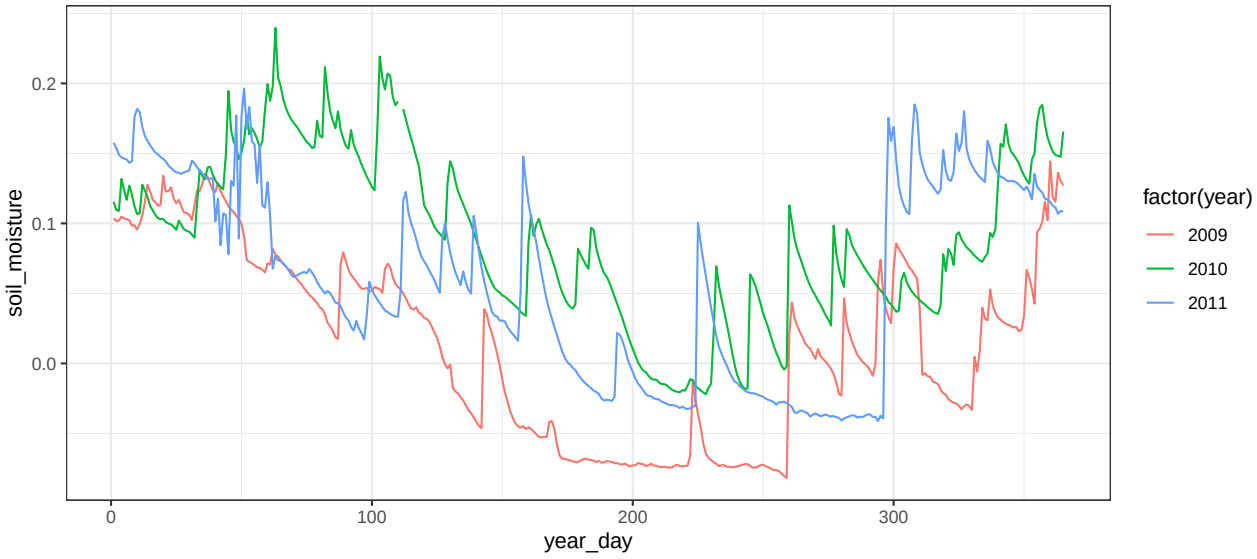
6.2 Plotting the daily average of a microsite for a given set of years

To compute the daily average of a given microsite we first have to subset the data relative to such microsite for the given set of years.

To compute the mean of the three sensors of the selected microsite, the data is grouped by the columns year and year_day, and after that dplyr::summarise() is used to compute the average of soil_moisture, but removing NA values from the computation of the average.

```
moiscrust %>%
 dplyr::filter(
  interpolated == FALSE,
  microsite == "stipa",
  year %in% c(2009, 2010, 2011)
```

```
) %>%
dplyr::group_by(year, year_day) %>%
dplyr::summarise(
  soil_moisture = mean(soil_moisture, na.rm = TRUE)
  ) %>%
ggplot2::ggplot() +
ggplot2::aes(
  x = year_day,
  y = soil_moisture,
  color = factor(year)
) +
ggplot2::geom_line() +
ggplot2::theme_bw()
```

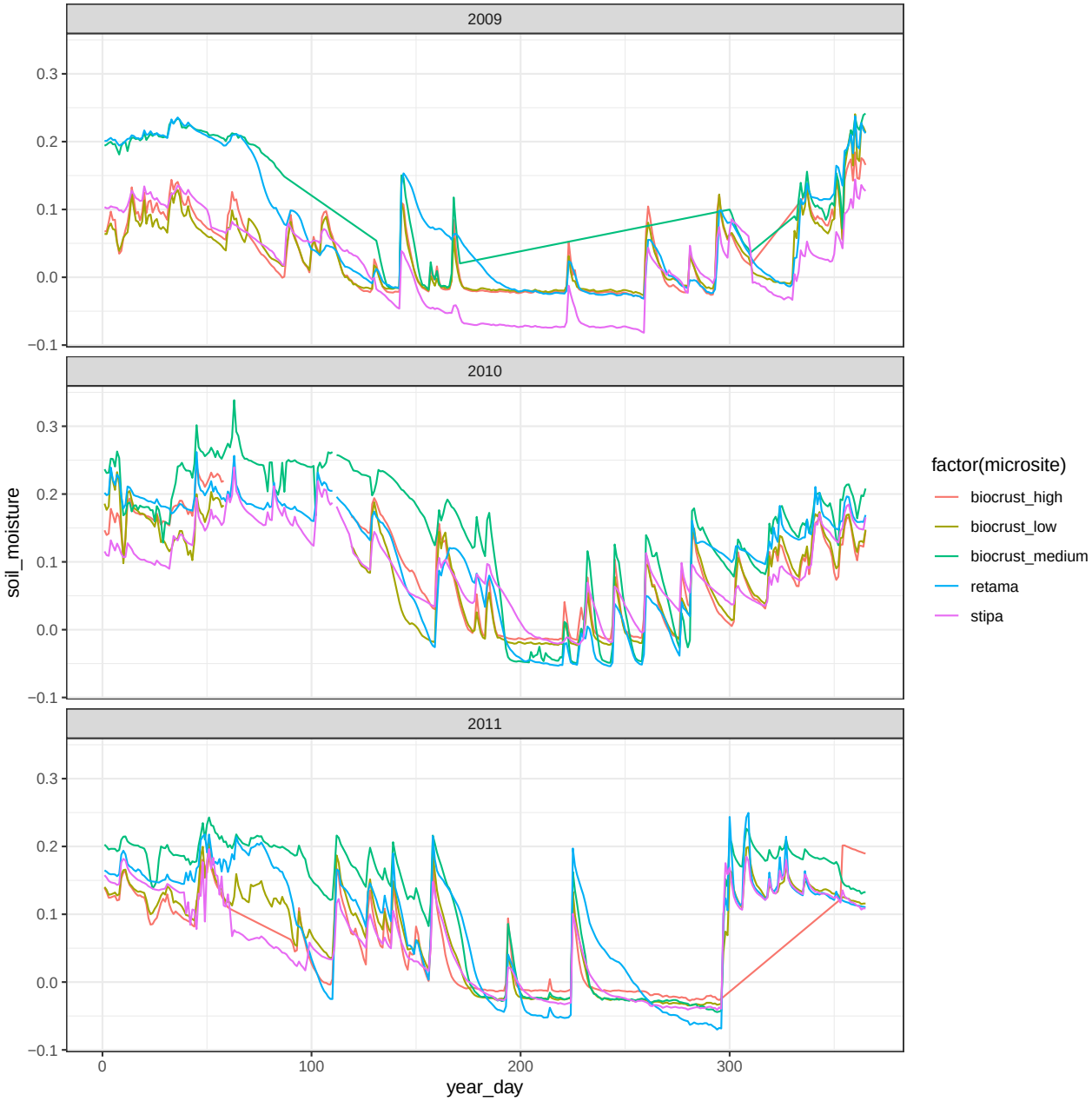


6.3 Plotting the daily average of each microsite for a given set of years

This case is very similar to the previous one, but here we remove the filter by microsite, and group by two variables instead: date_time_id and microsite.

```
moiscrust %>%
 dplyr::filter(
   interpolated == FALSE,
   year %in% c(2009, 2010, 2011)
 ) %>%
 dplyr::group_by(year, year_day, microsite) %>%
 dplyr::summarise(
   soil_moisture = mean(soil_moisture, na.rm = TRUE),
   ) %>%
```

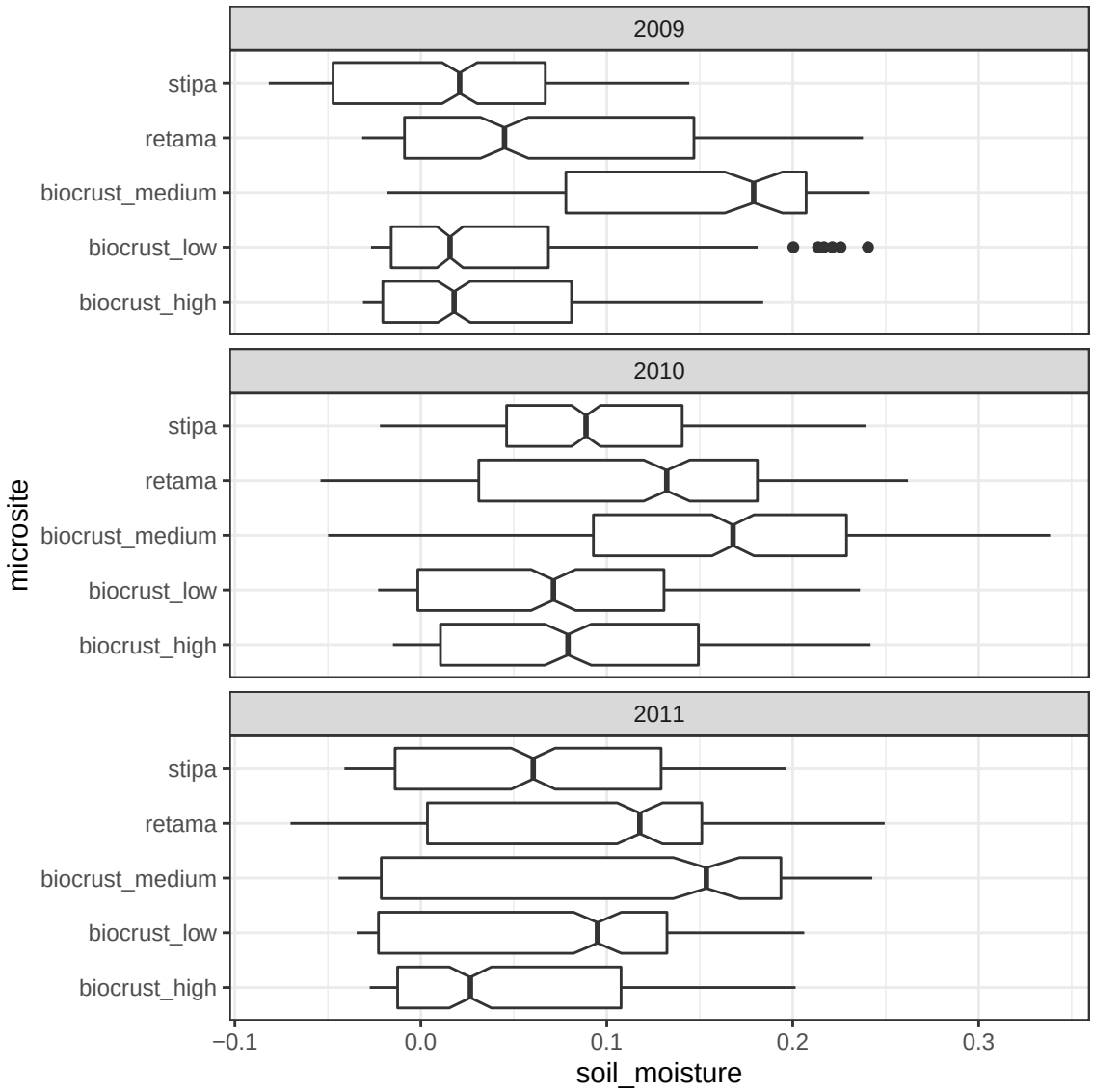
```
ggplot2::ggplot() +
ggplot2::facet_wrap("year", ncol = 1) +
ggplot2::aes(
  x = year_day,
  y = soil_moisture,
  color = factor(microsite)
) +
ggplot2::geom_line() +
ggplot2::theme_bw()
```



6.4 Comparing microsites across years

Changing the axes and replacing ggplot2::geom_line() with ggplot2::geom_boxplot(notch = TRUE) allows to compare the distributions of the soil moisture values between microsites across years.

```
moiscrust %>%
 dplyr::filter(
  interpolated == FALSE,
  year %in% c(2009, 2010, 2011)
 ) %>%
 dplyr::group_by(year, year_day, microsite) %>%
 dplyr::summarise(
  soil_moisture = mean(soil_moisture, na.rm = TRUE),
  ) %>%
 ggplot2::ggplot() +
 ggplot2::facet_wrap("year", ncol = 1) +
 ggplot2::aes(
  x = soil_moisture,
  y = microsite
 ) +
 ggplot2::geom_boxplot(notch = TRUE) +
 ggplot2::theme_bw()
```

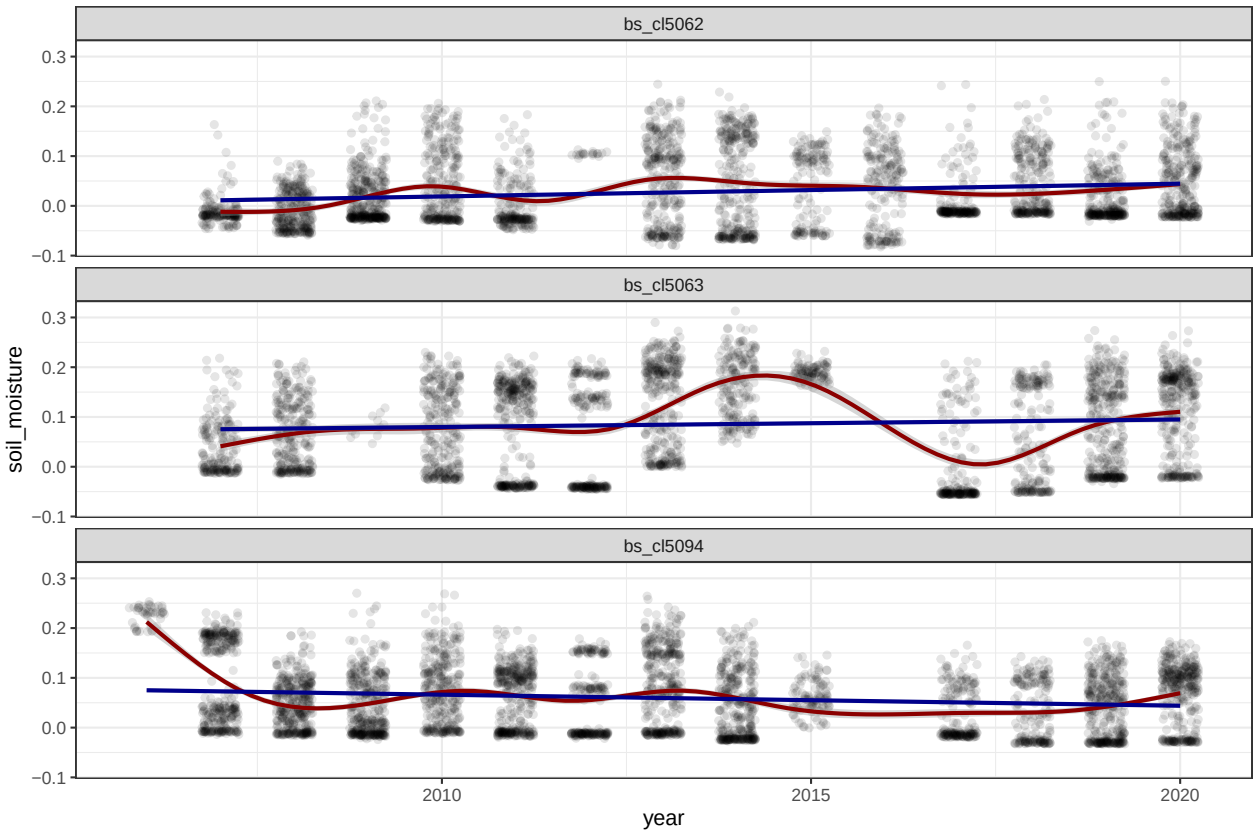


6.5 Graphical modeling of temporal trends

To model the temporal trends of the sensors (daily means over the years) for a given microsite the data must be grouped by year, year_day, and microsite. Adding a loess and a linear fit to the plot allows to quickly check temporal trends on each sensor at different time-scales.

```
moiscrust %>%
 dplyr::filter(
  interpolated == FALSE,
  microsite == "biocrust_low"
 ) %>%
 dplyr::group_by(year, year_day, sensor) %>%
 dplyr::summarise(
  soil_moisture = mean(soil_moisture, na.rm = TRUE),
```

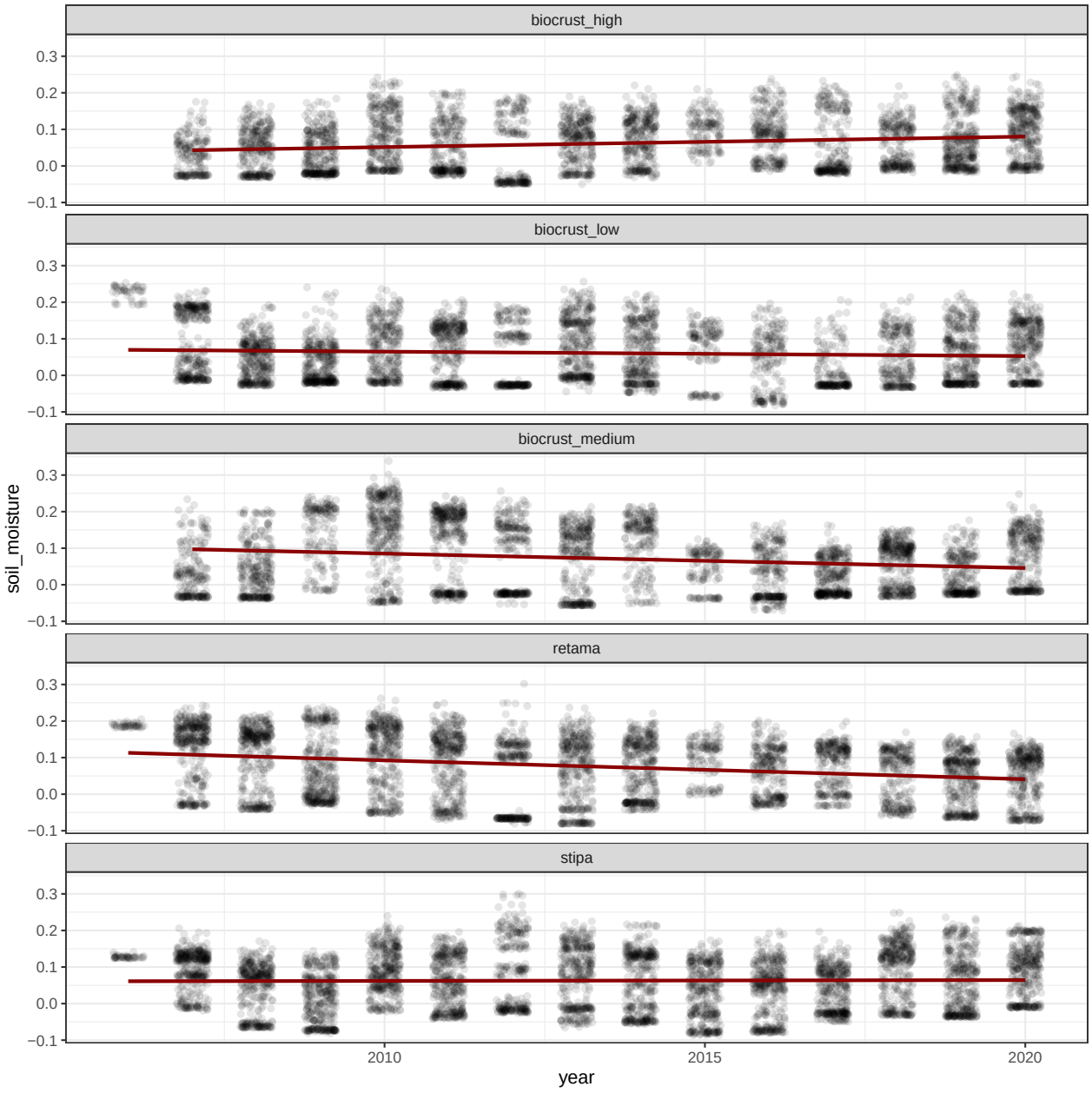
```
  ) %>%
 ggplot2::ggplot() +
 ggplot2::facet_wrap("sensor", ncol = 1) +
 ggplot2::aes(
  x = year,
  y = soil_moisture
 ) +
 ggplot2::geom_jitter(width = 0.25, alpha = 0.1) +
 ggplot2::geom_smooth(col = "red4") +
 ggplot2::geom_smooth(method = "lm", col = "blue4") +
 ggplot2::theme_bw()
```



6.6 Comparing temporal trends by microsite

```
moiscrust %>%
 dplyr::filter(
  interpolated == FALSE
 ) %>%
 dplyr::group_by(year, year_day, microsite) %>%
 dplyr::summarise(
```

```
  soil_moisture = mean(soil_moisture, na.rm = TRUE),
  ) %>%
ggplot2::ggplot() +
ggplot2::facet_wrap("microsite", ncol = 1) +
ggplot2::aes(
  x = year,
  y = soil_moisture
) +
ggplot2::geom_jitter(width = 0.25, alpha = 0.1) +
ggplot2::geom_smooth(method = "lm", col = "red4") +
ggplot2::theme_bw()
```


biocrust_high
biocrust_low
biocrust_medium
retama
stipa
soil_moisture
0.3
0.2
0.1
0.0
−0.1
2010
2015
2020
year