Methodology for working with time series of climate data and microclimatic maps

---

NmetS - Methodologies approved by the relevant state administration body under whose competence the given issue falls

This methodology is a result of the project: **Forest microclimate in time and space: real impacts of climate change on selected protected areas**

Project number: **SS06010011**

Project start: **04/2023**

Project end: **03/2026**

Project investigator: Institute of Botany of the Czech Academy of Sciences, Zámek 1, Průhonice, 25243, Czech Republic

Confidentiality and availability: S – Complete and truthful project data not subject to protection under special legal regulations.

Original release: **2024**

Version: **1.1**

Updated: 22.5.2025 (Update to myClim 1.40)

Tested with R 4.4.1, myClim 1.40, RStudio 2025.05.0

English version was translated by AI - Claude Sonnet 3.5 and manual checking was not done yet.

T A
Č R
Programme **Environment For Life**

This project is funded with state support from the Technology Agency of the Czech Republic and the Ministry of the Environment of the Czech Republic under the Environment for Life Program.

Information about the author team:

RNDr. Josef Brůna, Ph.D., Principal Investigator

Mgr. Tereza Klinerová, Research Team Member

Mgr. Martin Kopecký, Ph.D., Research Team Member

Mgr. Martin Macek, Ph.D., Research Team Member

Mgr. Matěj Man, Research Team Member

Mgr. Anna Růžičková, Research Team Member

doc. Ing. Jan Wild, Ph.D., Research Team Member

This project is funded with state support from the Technology Agency of the Czech Republic and the Ministry of the Environment of the Czech Republic under the Environment for Life Program.

Contents

Introduction

Microclimatic measurements are now widely available thanks to the development of automated autonomous measuring instruments, but their processing can be challenging due to large data volumes and non-standardized output data formats as well as non-standardized measurement methods depending on the specific research question. Therefore, they are only occasionally used in nature conservation (e.g., when assessing the impact of climate on protected features). The aim of this methodology is to facilitate the use of microclimatic data in nature conservation practice and ensure reproducibility of results through practical examples and proven procedures. The target user group is primarily staff of National Park and Protected Landscape Area administrations, as well as staff of the Nature Conservation Agency.

The first part of the methodology focuses on deriving biologically relevant variables, particularly based on time series from TOMST microclimatic stations, but also from other sources, so that other compatible data can be used as well. The main motivation is the fact that while classical meteorological data is already standardized and offers a number of comparable variables available from different sources, data from compact microclimatic stations is far from being as standardized and analysis of data obtained from individual stations can be challenging at first glance. Using examples based on real data, we will show the main problems in data processing and recommend proven procedures. The second part of the methodology focuses on the practical use of derived variables for data evaluation and also the use of microclimatic and climatic maps.

The methodology describes procedures for obtaining and processing microclimatic measurements from autonomous dataloggers (installation, data management, data quality control, processing and visualization of measured values, basic data analysis) completely in the open-source R programming environment. We believe that the absence of ties to commercial software will facilitate its use. Another advantage of processing data in R is the reproducibility of procedures and results, as all data operations remain documented in the source code.

How to work with the methodology

This methodology exists as a standalone document (in .pdf format), but is also available online including all source files used in the data processing examples. The entire project for RStudio - https://www.rstudio.org/ and the individual tasks presented in it can thus be run by users and the code can be modified for their own needs on their device.

The .Rmd format combines the actual document text and executable R code; for custom analyses, individual parts of the code can be transferred to a new script in R format. At the same time, the codes of individual parts of the methodology are available for the R environment without the surrounding text (.R format).

Due to the dynamic nature of R libraries, the online version of the methodology will be continuously updated during project implementation to be compatible with the current development of the *myClim* library (Man et al. 2023) and other important libraries and data sources. Minor deviations may still occur and cause errors in code execution; please do not hesitate to contact us if needed.

The online version with all sample data is available here: https://gitlab.ibot.cas.cz/vysledky/microclimate-methodology

From there, you need to download from the web using the **Code** button - select **zip** here. The direct link is: https://gitlab.ibot.cas.cz/vysledky/microclimate-methodology/-/archive/main/microclimate-methodology-main.zip

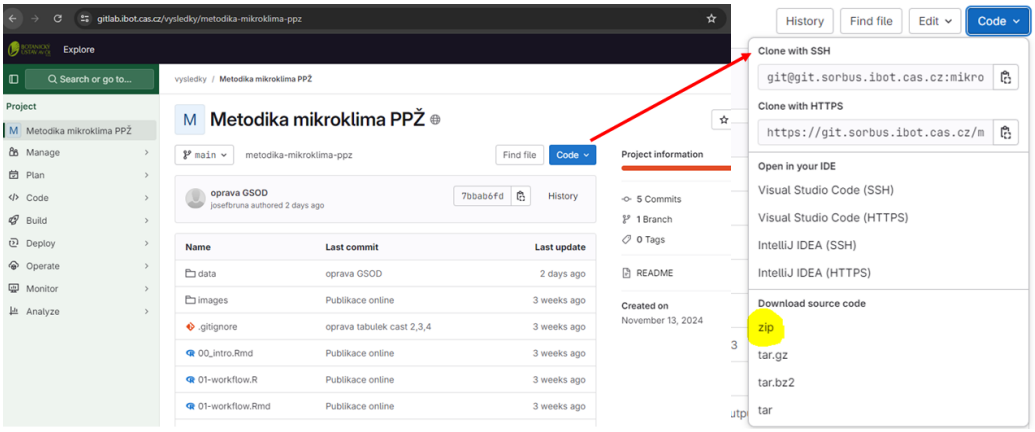

Figure 1: Downloading the repository in ZIP format.

Then extract the contents on your disk and open the microclimate-methodology.Rproj file in RStudio.

In the RStudio window, you can then open individual **Rmd** files (**Figure 2**, in the files section at the bottom right) and run the code in the main window using the green arrows in the top right corner of each code window:

We recommend displaying with formatting using the **Visual** button. In the section with variables in memory, you can find currently loaded variables and examine their contents. Navigation is facilitated by an outline that can be turned on in the right part of the main window using the **Outline** button. At the bottom left is the console output, where you can see the progress of individual commands and some less important outputs may be displayed, as well as error messages.

The main outputs (images) are always below each code block (**Figure 3**). Individual chapters need to be run sequentially from top to bottom, as subsequent code blocks often use variables created in previous blocks.

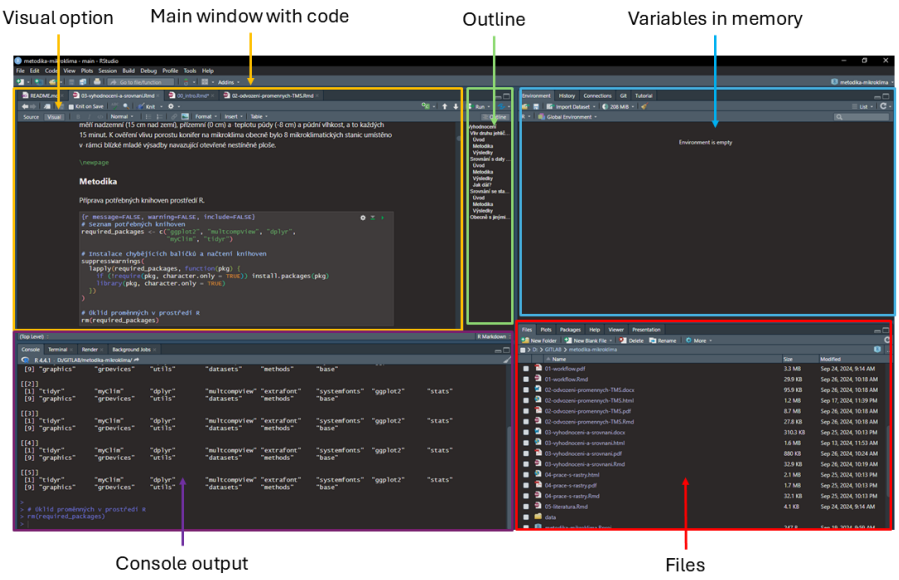

Figure 2: RStudio environment with labeled parts, RStudio's color scheme and window layout may vary.

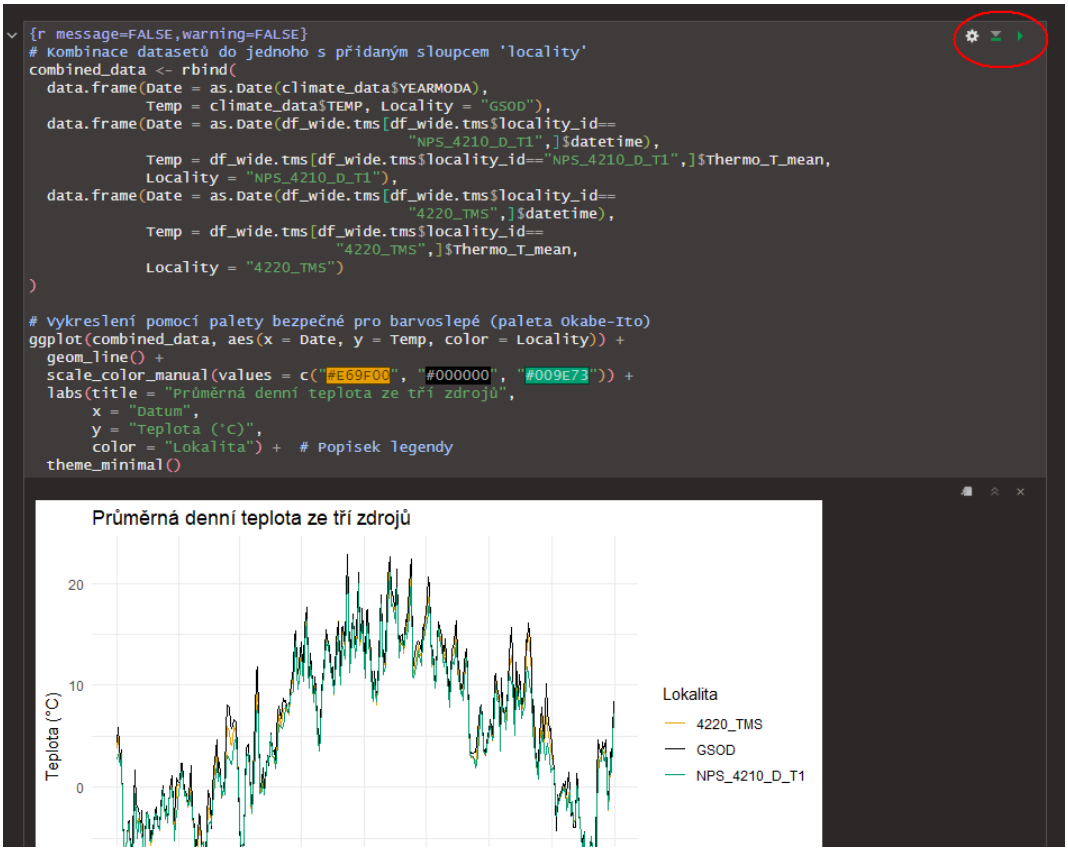

Figure 3: Example output of one code block, the code block run button is highlighted in red.

Environment Setup and Functions for Table Rendering

1. Workflow

In this initial section, we will focus on obtaining relevant microclimatic data and preparing it for input into subsequent analyses. We will go through the entire process from installing sensors in the field, through data downloading to their control, cleaning, calibration, and joining time series from multiple readings. We will also show how to keep records of sensor history (e.g., readings, sensor exchanges within a location) and set up an appropriate structure for storing downloaded data.

We emphasize working with TMS (TOMST) system sensors - https://tomst.com/web/en/systems/tms/, currently widely used worldwide for microclimatic measurements (Lembrechts et al., 2022), but the methodology can be generalized to other sources of microclimatic data - e.g., from instrumental measurements by HOBO (Onset), iButton (Maxim/Dallas), Minikin (EMS Brno), EnvLoggers (ElectricBlue) and others. For working with microclimatic data, we use the R library *myClim* (Man et al., 2023), in which we have gathered experience from many years of working with microclimatic data.

1.1. Installation of TMS Sensors in the Field

Proper installation of TMS sensors in the field is a basic prerequisite for obtaining relevant microclimatic data. It is preceded by the selection of representative locations for sensor placement, which depends on the specific research question. Following uniform procedures ensures the comparability of measured data. Detailed information is summarized in the *Methodology for measuring microclimate using TMS system microclimatic stations* (Brůna et al., 2021); here we provide only a brief summary.

Thermologger The Thermologger (TOMST) sensor is an autonomous datalogger measuring air temperature. It is mounted using an M5 metric thread into white shielding, protecting the sensor from direct sunlight, typically in a vertical position at a height of 2 m above ground (unless the nature of the research question requires different placement). In forest stands, it can be placed on the north side of a tree trunk **(Fig. 4 A and B)** or on an inexpensive and durable plastic-coated garden stake.

TMS-4 datalogger The TMS-4 datalogger (TOMST) was designed for monitoring microclimatic variables relevant to the herb layer (Wild et al., 2019). The TMS-4 datalogger measures temperature at three positions - air temperature 15 cm above ground, soil surface temperature, soil temperature 8 cm below soil surface, and soil moisture. The sensor is typically installed in a protective cage that prevents damage by wildlife and falling branches **(Fig. 4 E)**, with the signal LED facing south; the correct burial depth of the lower (green) part of the datalogger with the soil moisture and temperature sensor is shown in **Fig. 4 C**. If the sensor is installed in stony soil, it is advisable to use an auxiliary metal tool to dig the hole to avoid scratching or completely damaging the soil moisture sensor (circuit on the PCB). It is important to check that the entire lower part of the sensor is in good contact with the soil (the sensor holds firmly in the dug hole, does not wobble) and to fill any air pockets with soil substrate. Single or double shielding can be used to shield the sensor from direct sunlight. Two variants of sensor placement on the research plot are shown in **Fig. 4 D** (Thermologger on tree) and **Fig. 4 F** (Thermologger on plastic-coated pole).

1.2. Data Download and Management

The interface for sensor operation (**Fig. 5**) - data downloading and basic data view is provided by the *Lolly* software from the TMS sensor manufacturer. During first installation, you need to have a TMD adapter connected to your computer. A complete guide for operating the software can be found on the manufacturer's website: https://tomst.com/web/wp-content/uploads/2023/06/Lolly-software-Handbook.pdf (in English).

From a practical use perspective, we particularly point out:

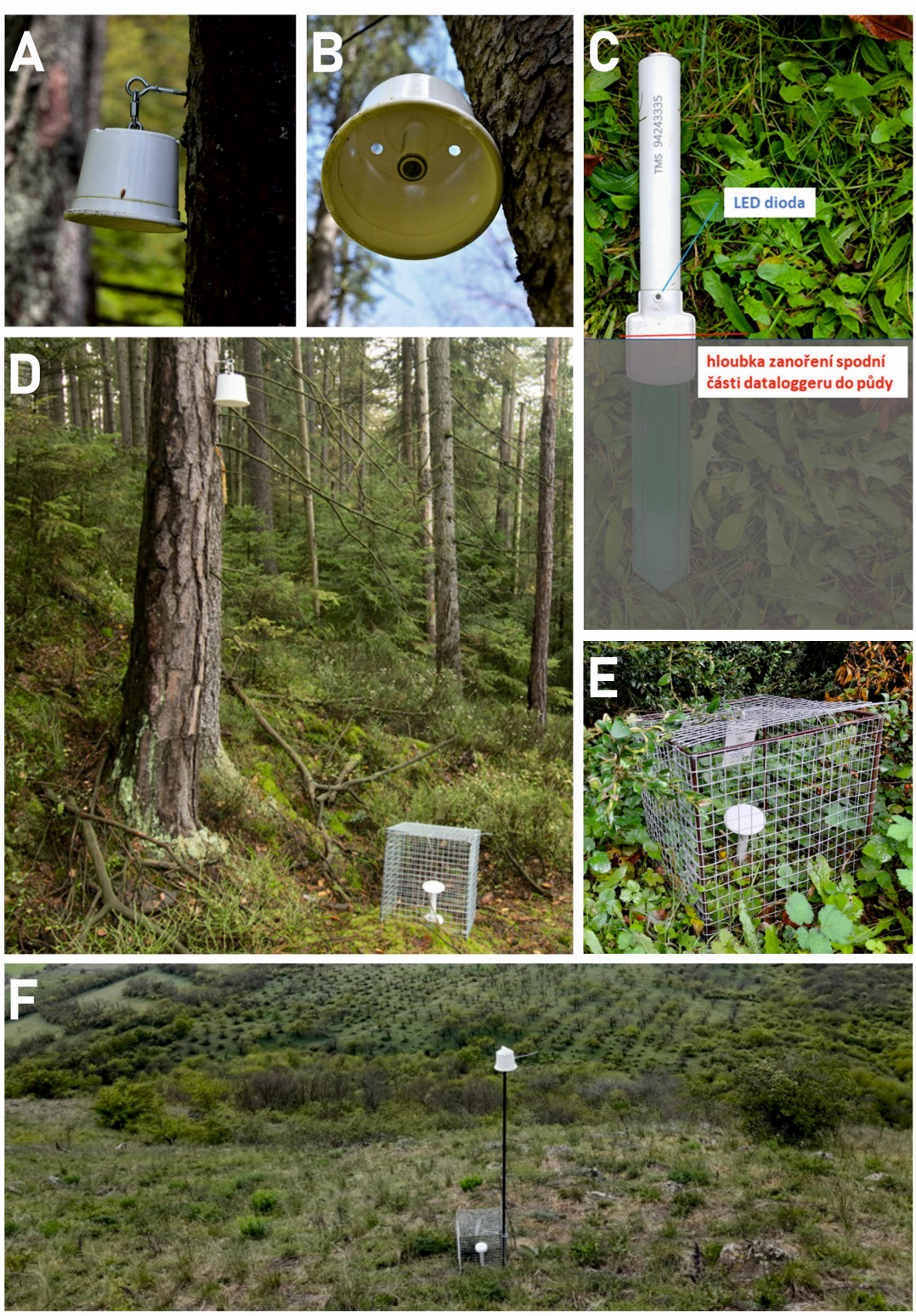

Figure 4: A) and B) Thermologger mounted on the north side of a tree trunk 2 m above ground. C) TMS-4 datalogger - location of the signal LED (points south when installed in the field) and correct burial depth of the lower (green) part of the datalogger in soil. D) Research plot equipped with TMS-4 datalogger and Thermologger mounted on a tree trunk. E) TMS-4 datalogger in protective cage. F) Research plot equipped with TMS-4 datalogger and Thermologger mounted on a pole.

- ability to set different measurement intervals
- ability to set different date and time formats (pay attention to compatibility with *myClim*)
- ability to create a bookmark
- ability to read data from a specific date or bookmark
- ability to display an overview graph of measured data

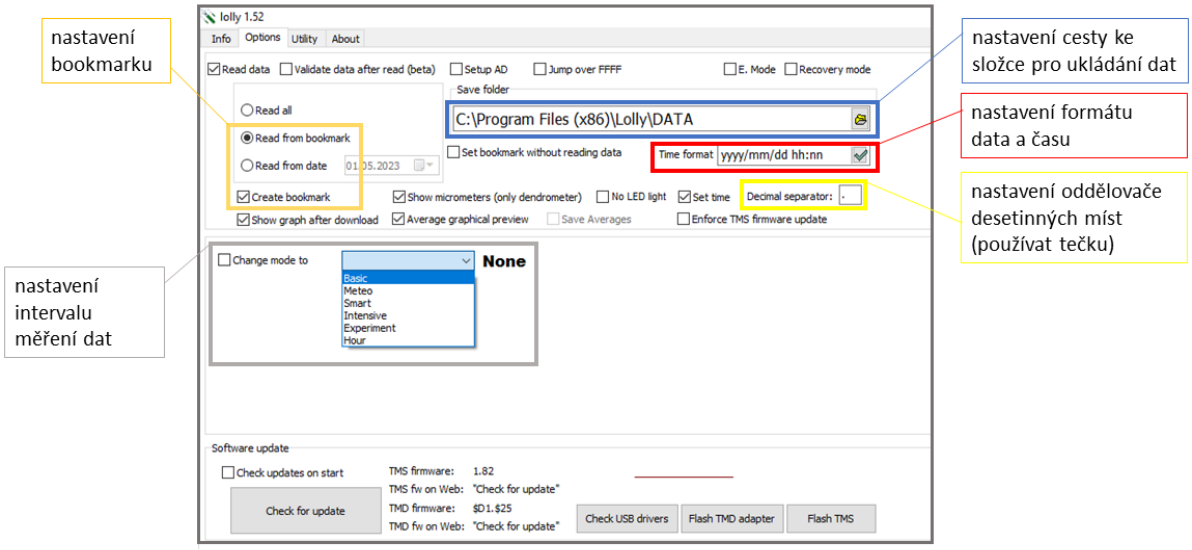

Figure 5: Interface of *Lolly* software - setting parameters often needed in practice. Complete guide for operating the software is here: tomst.com/web/wp-content/uploads/2023/06/Lolly-software-Handbook.pdf.

Time series with measured values are stored after reading the datalogger into a CSV file named with the text string:

"data_" + serial number + year_month_day + reading sequence number + ".csv"

The file contains tabular data in the form of semicolon-separated values. The individual columns represent sequentially: observation sequence number, measurement date and time in UTC, time zone code, temperature sensor T1, temperature sensor T2, temperature sensor T3, soil moisture, shock indicator, error flag. For TOMST TMS-4 loggers, T1 is the soil sensor (-8cm), T2 is the ground sensor (+2cm), T3 is the air sensor (+15cm). For TOMST Thermologger, temperature is stored in field T1, other unused fields contain value -200.

Additionally, a "command_*.csv" file containing a log of the reading process is stored along with the data, which we won't need for further work, but we recommend archiving it for possible troubleshooting with the manufacturer.

It is advisable to store downloaded data on cloud storage for online data backup (ability to set the path for storing data in a synchronized folder directly in the *Lolly* software). It is appropriate to store data in folders so that it is subsequently easy to link field notes with the data storage. For example, if we read data once a year, then name folders according to the given year and organize notes in field records also with the tag of the given year. We archive a copy of downloaded files preferably in a compressed archive (.zip) to save space, as .csv files are data intensive.

If you need to change the date and time format retrospectively for already read data, you can use the *Lolly* software (unofficial guide see **Fig. 6 A-C**).

Field data collection should be done once or twice a year (spring, autumn) to prevent larger data loss in case of sensor failure. For routinization and easier coordination of reading work, you can use the "field

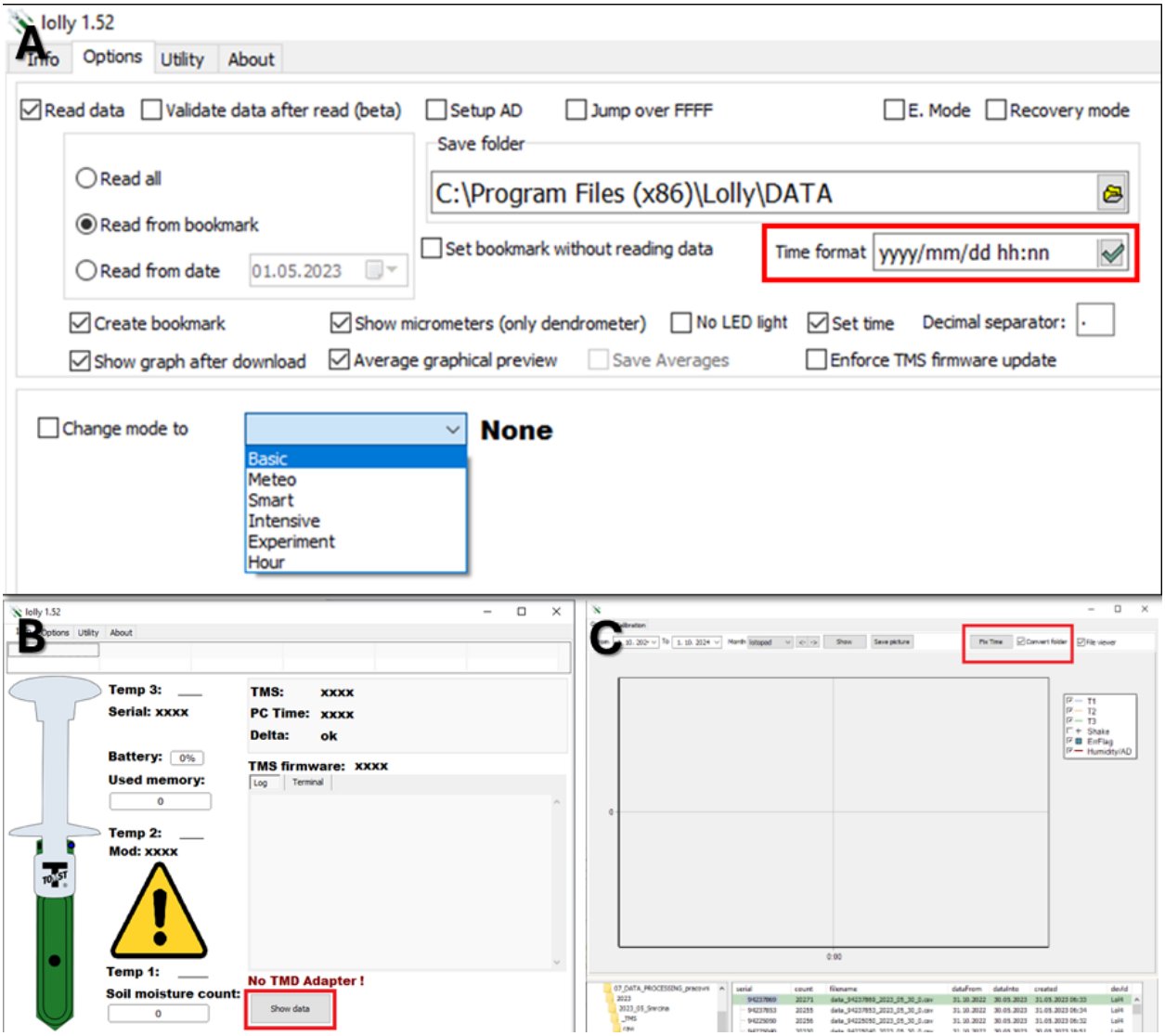

Figure 6: A) First set the desired date format compatible with *myClim* software on the Options tab. B) Then go to the Info tab and click on "Show Data". C) This activates the file browser, navigate to the folder where you have stored data intended for conversion to a different date and time format, and click on "Fix time". This will convert all .csv files in the given folder to the date and time format set on the Options tab. Converted files will have the prefix *conv_*.

manual" given in Appendix 1 of the above-mentioned *Methodology for measuring microclimate using TMS system microclimatic stations* (Brůna et al., 2021). It is necessary to record the progress of reading and all non-standard events (sensor pulling out, fallen shielding, ID of newly installed sensor, etc.). These notes are essential for proper data processing:

a) to check whether we have readings from all sensors and locations,
b) for correct interpretation of data and evaluation of possible errors in data.

Common Problems and Failures Detectable Directly in the Field

- **poor contact of the lower part of TMS-4 sensor with soil**

If the sensor is loose in the soil, there is a risk of inaccurate soil moisture measurement due to poor contact with soil (sensor measures only in a small volume of soil closely adhering to the sensor). In the graph, the problem manifests as sudden drops in moisture, or generally low values (below 1000 units). The sensor needs to be reinstalled. We note the problem and during processing of measured values we exercise increased caution regarding soil moisture, possibly replacing soil moisture values with NA.

- **pulled out TMS-4 sensor**

If a TMS-4 sensor is found lying freely on the soil surface (and is not visibly mechanically damaged), check the functionality of the sensor in *Lolly* software (no measurement dropouts, constant or obviously nonsensical values of measured variables are visible in the overview graph). If the sensor is functional, reinstall it. However, when processing data, it is necessary to find out when the sensor was pulled out of the soil, which can be recognized by a sudden equalization of temperatures T1, T2 and T3 and a sharp drop in soil moisture values. In that period, we must then replace all data series with NA values (temperature T2 measured at the soil surface can be preserved with a liberal approach), more details below. The *myClim* library also offers a function for automatic detection of data measured on a pulled out sensor `mc_prep_TMSoffsoil()`.

- **fallen shielding**

Especially TMS sensors without wildlife protection suffer from loss of shielding. The shielding needs to be returned to the sensor. In such a case, the measured temperatures are not comparable with values measured on shielded sensors (especially temperature 15 cm above ground and temperature at soil surface). Shielding prevents sensor overheating when direct sunlight hits the sensor.

- **sensor failures**

Recognizable in the overview graph in *Lolly* software. If these are one-time events, sensor functionality may not be limited and we can leave it in place. If the failures are more extensive, replace the sensor with a new one. If an entire time period is missing, it may be a data transfer error during reading. In such a case, repeat the reading. If reading from bookmark is set, it is necessary to temporarily change this option and read the sensor entirely or from a specific date, because a new bookmark has probably already been created.

- **time shift**

During each reading, the time in the computer and in the sensor is synchronized. If the internal time on the sensor and in the computer significantly disagree (i.e., difference more than 10 minutes shown in *Lolly* in the "Delta:" field - **Figure 5B**), this fact needs to be noted in field records and find out the probable cause. It may be due to lagging internal datalogger clock, e.g., in sensors with weakening battery. It may also be an error due to incorrectly set system time on the computer either during the previous reading or during the current reading (note: TMS sensors show UTC time, so they differ from Central European time by 1 hour, or by 2 hours from summer time).

1.3. Record Keeping

A necessary condition for preparing relevant input data is records about sensor placement at the location and notes from field readings, which serve for correct interpretation and evaluation of data. During site visits, it is therefore necessary to record all changes in sensor status. This information is then an integral part of subsequent processing. The simplest example of such a record table for our sample data might be this:

```
# Loading field notes from Excel
zaznamy <- read.xlsx("./data/zaznamyR.xlsx")
```

Table 1: Example of field notes

locality_id	logger_type	2021	2021.note	2022	2022.note	2023	2023.note
CZ2_MASTALE	HOBO_RH	20024371	ok, 30min	20024371	ok, 30min	20024371	membrane replaced
CZ2_HODDUB	HOBO_RH	21061738	ok, 30min	21061738	ok, 30min	21061738	ok, 30min
CZ2_CEPICKA	HOBO_RH	21061739	ok, 30min	21061739	ok, 30min	21061739	ok, 30min
CZ2_LUZNICE	HOBO_RH	21061759	ok, 30min	21061759	ok, 30min	21061759	ok, 30min
CZ2_KLINOV	HOBO_RH	21061889	ok, 30min	21061889	ok, 30min	21061889	ok, 30min
CZ2_HRUSUT	HOBO_RH	21061891	ok, 30min	21061891	ok, 30min	21061891	ok, 30min
CZ2_BARINY	HOBO_RH	21067836	ok, 30min	21067836	ok, 30min	21067836	membrane replaced
CZ2_KRKLAB	HOBO_RH	21067844	ok, 30min	21067844	ok, 30min	21067844	ok, 30min
CZ2_BUKOVEC	HOBO_RH	21067850	ok, 30min	21067850	ok, 30min	21067850	ok, 30min
CZ2_HODDUB	Thermo	91171001	ok	91171001	ok	91171001	ok
CZ2_LUZNICE	Thermo	91171050	ok	91171050	1x error	91171050	ok
CZ2_CEPICKA	Thermo	91192386	ok	91192386	ok	91192386	ok
CZ2_BUKOVEC	Thermo	91201102	ok	91201102	ok	91201102	ok
CZ2_MASTALE	Thermo	91201109	ok	91201109	ok	91201109	ok
CZ2_HRUSUT	Thermo	91201114	ok	91201114	ok	91201114	ok
CZ2_BARINY	Thermo	91201121	ok	91201121	ok	91201121	ok
CZ2_KLINOV	Thermo	91201144	ok	91201144	ok	91201144	ok
CZ2_KRKLAB	Thermo	91201145	ok	91201145	ok	91201145	ok
CS_26	TMS4	94183144	ok	94183144	out of soil	94183144	out of soil, chewed, downloaded
CZ2_CEPICKA	TMS4	94192751	ok	94192751	ok	94192751	moisture error, -5:02:21

For each reading (in the case of our sample data, readings once per year), it is necessary to record the datalogger serial number (ID) (printed on the tube), unique location identifier, and other notes about the reading for unambiguous identification. In our example, the same dataloggers are located at the sites permanently, but this may not be the rule - if damage occurs or the datalogger is replaced over time, the ID changes. Thanks to these records, we will later be able to correctly assign data series to individual locations, even if the datalogger is replaced. We also strongly discourage renaming files that are identified by the datalogger serial number, because the datalogger serial number also serves to assign calibration data. From the reading notes, it is clear that when processing data, we will need to focus on dataloggers at locations CS_26, where the datalogger was pulled out of the ground, which invalidated the measurements, and we will also focus on locations CZ2_CEPICKA and CZ2_LUZNICE, where we have a note about moisture sensor failure.

1.4. Loading and Cleaning Raw Microclimatic Data

Example of data preparation for further analysis from raw field data. As an example, we present data measured at 10 locations across the Czech Republic. At all locations, TOMST TMS-4 dataloggers were placed measuring air temperature at heights of 15 cm and 2 cm above soil surface, 8 cm below soil surface, and soil moisture in the top soil layer (0 - 15 cm). At 9 locations, there was also a Thermodatalogger measuring air temperature 200 cm above ground and a HOBO U23 datalogger measuring air temperature and relative humidity 150 cm above ground. The time series contains measurements from approximately

April 2021 to October 2023, but not all dataloggers measured for the same duration, there are missing segments in the data.

```
# List of required libraries
required_packages <-
  c("myClim", "dplyr", "tidyr", "stringr", "openxlsx", "ggplot2")

# Install and load libraries
suppressWarnings(lapply(required_packages, function(pkg) {
  if (!require(pkg, character.only = TRUE))
    install.packages(pkg, repos = "https://mirrors.nic.cz/R/")
  library(pkg, character.only = TRUE)
}))

# Clean up variables in R environment
rm(required_packages)
```

We have installed and loaded the necessary libraries for working with microclimatic data and now we will load information about files read in the field. For data processing in the *myClim* library, it is important to correctly assign individual files to locations, correctly enter the expected data format (type of datalogger from which the data comes), and further define in what format dates and times are given in the files (whether it is e.g., 2023.12.31 12:00 or 31.12.2023 12:00:00). We call this definition file **files_table,** we can create it in code from our record table or manually in a spreadsheet editor and then load it in CSV format.

```
## assembly of files_table from the above loaded excel table "zaznamy"
## in combination with paths to downloaded files from loggers
## Getting paths to downloaded files
soubory <- list.files("./data/example_data_prep_calc",
                      recursive = T,
                      full.names = T)
## Extraction of ID for 2 types of loggers from file names (TMS, HOBO)
jmena <- basename(soubory) # file names
logger_id <-
  str_match(jmena, "data_\\s*(.*?)\\s*_") # extraction of TOMST TMS ID
hob <-
  is.na(logger_id[, 2]) # which loggers don't have TMS ID, those are HOBO
logger_id[hob] <-
  substr(jmena[hob], 1, 8) # extraction of HOBO logger IDs

## Extraction of reading year from folder name where data is located
odecty <- basename(dirname(soubory))

## Now we know the path, logger ID and reading year.
ft <- data.frame(path = soubory,
                 logger_id = logger_id[, 2],
                 odecty = odecty)

## However, we still lack information about location, logger type and date format
## we get this information from the field records table
## we need a key of which loggers were at which location in which year
## these are columns 3,5,7
zaznamy.dlouha <- pivot_longer(zaznamy,
  cols = c(3, 5, 7),
  names_to = "odecty",
```

```
  values_to = "logger_id")

## columns with notes are not needed now
## we get a table with key locality-logger_type-reading-logger_id
zaznamy.dlouha <- zaznamy.dlouha[,-c(3:5)]

## create "files_table"
## joining location key and path to files by logger ID and reading year
files_table <-
  merge(ft, zaznamy.dlouha, by = c("logger_id", "odecty"))

## derive myClim data type from logger type information
data_format <- files_table$logger_type
data_format[data_format == "Thermo"]  <- "TOMST"
data_format[data_format == "TMS4"]    <- "TOMST"
data_format[data_format == "HOBO_RH"] <- "HOBO"
files_table$data_format <- data_format

## by examining date columns in our logger files
## we find that HOBO files have a consistent date format
## TOMST files have 4 date formats
## date format depends on reading software settings
## the more people and computers involved in downloading data, the more possible formats
date_format <- files_table$data_format
date_format[date_format != "HOBO"] <-
  "%d.%m.%Y %H:%M:%S@%d.%m.%Y %H:%M@%Y.%m.%d %H:%M@%d.%m.%Y"
date_format[date_format == "HOBO"] <- "%d.%m.%Y %H:%M:%S"
files_table$date_format <- date_format

## Keep only necessary columns
files_table <-
  files_table[, c("path", "locality_id", "data_format", "date_format")]
```

The files_table.csv file must follow the given structure (see Table 2):

Table 2: Example structure of table defining file association with locations and file data type

path	locality_id	data_format	date_format
./data/example_data_prep_calc/2021/20024371.txt	CZ2_MASTALE	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2022/20024371.txt	CZ2_MASTALE	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2023/20024371.txt	CZ2_MASTALE	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2021/21061738.txt	CZ2_HODDUB	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2022/21061738.txt	CZ2_HODDUB	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2023/21061738.txt	CZ2_HODDUB	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2021/21061739.txt	CZ2_CEPICKA	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2022/21061739.txt	CZ2_CEPICKA	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2023/21061739.txt	CZ2_CEPICKA	HOBO	%d.%m.%Y %H:%M:%S
./data/example_data_prep_calc/2021/21061759.txt	CZ2_LUZNICE	HOBO	%d.%m.%Y %H:%M:%S

Now we will load the actual data from microclimatic stations into one *myClim* object (raw format):

```
## Loading microclimatic data into myClim
## assigning loggers to locations according to "files_table"
micro.data <-  mc_read_data(files_table = files_table,
```

```
                silent = F,
                clean = F)

## Loading using files_table prepared manually, saved to .csv file
micro.data <-  mc_read_data(files_table = "./data/files_table.csv",
                silent = F,
                clean = F)

## Perform basic machine cleaning and data checking, time step consistency
## duplicates, time axis sequence, missing values
micro.data.clean <- mc_prep_clean(micro.data, silent = T)
```

For a basic overview of the data, we will print out information about data range, measurement step and errors in data (missing values, duplicates, wrong ordering).

```
## Printing information about loggers: data range, measurement step,
## duplicates, order, missing data
info.tms <- mc_info_clean(micro.data.clean)
```

Table 3: Result of data check in object 'micro.data.clean' - measurement range, counts of duplicate, missing and chronologically incorrectly ordered records.

locality_id	serial_number	start_date	end_date	step	duplicities	missing	disordered	rounded	NA
CS_26	94183144	TMS_1	2021-04-09	2021-11-03	900	0	0	0	FALSE
CS_26	94183144	TMS_2	2021-11-03	2022-10-19	900	0	0	0	FALSE
CS_26	94183144	TMS_3	2022-10-19	2023-04-06	900	0	0	0	FALSE
CZ2_BARINY	21067836	HOBO_U23-001A_1	2021-04-26	2021-11-08	1800	0	0	0	FALSE
CZ2_BARINY	21067836	HOBO_U23-001A_2	2021-11-08	2022-10-24	1800	0	0	0	FALSE
CZ2_BARINY	21067836	HOBO_U23-001A_3	2022-10-24	2023-10-31	1800	0	0	0	FALSE
CZ2_BARINY	91201121	Thermo_1	2021-04-29	2021-11-08	900	0	0	0	FALSE
CZ2_BARINY	91201121	Thermo_2	2021-11-08	2022-10-24	900	33602	0	1	FALSE
CZ2_BARINY	91201121	Thermo_3	2022-10-24	2023-10-31	900	0	0	0	FALSE
CZ2_BARINY	94196117	TMS_1	2021-04-29	2021-11-08	900	0	0	0	FALSE

We are working with data from autumn 2021 to autumn 2023 measured at 10 locations across the Czech Republic. These are data from three different reading actions. The protocol printout from machine control from the *myClim* library `mc_info_clean()` shows a concerning number of duplicate values and values in incorrect order in some files. Upon closer examination of individual files, we will see that parts of the time series are indeed repeated in .csv files for certain dataloggers. This is a known bug occurring in data read using an older version of the *Lolly* reading software. We demonstrate such "damaged" files here intentionally. No measurements are missing in the data, on the contrary, there are extra ones. Without machine control `mc_prep_clean()` it would be very difficult to detect this problem. Similarly with incorrect ordering of consecutive measurements on the time axis.

If one measurement is missing in the data or there are a few duplicates in the data, it is usually an error that occurs during reading during the season when time is reset, and resolving these errors can be left to automatic correction by *myClim* function `mc_prep_clean()`.

Larger numbers of errors in data usually indicate a real datalogger failure or reading software problem or other issue and require deeper inspection and individual manual resolution at the level of source .csv files - erroneous values in the source file are manually replaced with NA or we use the *myClim* function `mc_states_insert()` which allows assigning a certain flag (tag) to a specific time segment of measurement of a given sensor and subsequently using the command `mc_states_replace()` replace all segments marked with a certain tag with NA.

In our case, it is therefore necessary to open files with a high number of errors in a spreadsheet editor and examine them. It turns out that these are not errors that would prevent the use of data in analysis, we can trust the machine control of *myClim* here, which will remove duplicate values and ensure correct sequence of the time series.

In addition to the `mc_info_clean()` printout, visual inspection of graphical display of data will help us. Either using static images, or interactively in the *myClimGUI* application (see below). For an overall overview of data progression, we can use the functions `mc_plot_lines()` **(Fig. 7)** and `mc_plot_raster()` **(Fig. 8)**. Both functions are optimized both for writing resulting images to PC disk and for viewing in R.

In the table of field records we see problems (kicked out datalogger) at location CS_26. For example, let's display the problematic location together with another, problem-free location. Among other things, on the line graph **(Fig. 7)** we see that the time series consists of different readings (files), which will be shown to us by the parameter `color_by_logger = TRUE`. The anomaly can also be seen on the raster graph **(Fig. 8)**. One of the next steps in preparing data for analysis will be joining records into one time series. We also see a sharp drop in moisture at location CS_26, which suggests that the datalogger got out of contact with soil, as we already know from field notes.

```
## Static display of temperature and soil moisture progression of TOMST TMS-4 logger
## Line graph with legend adjusted to 3 rows
# Use guides() and guide_legend() to set number of rows in legend
mc_plot_line(micro.data.clean[c("CS_26","CZ2_LUZNICE")], # Select locations
             sensors = c("TMS_T3","TMS_moist"), # Use specific sensors
             color_by_logger = TRUE) + ggplot2::guides(color
                                    = ggplot2::guide_legend(nrow = 3))
```

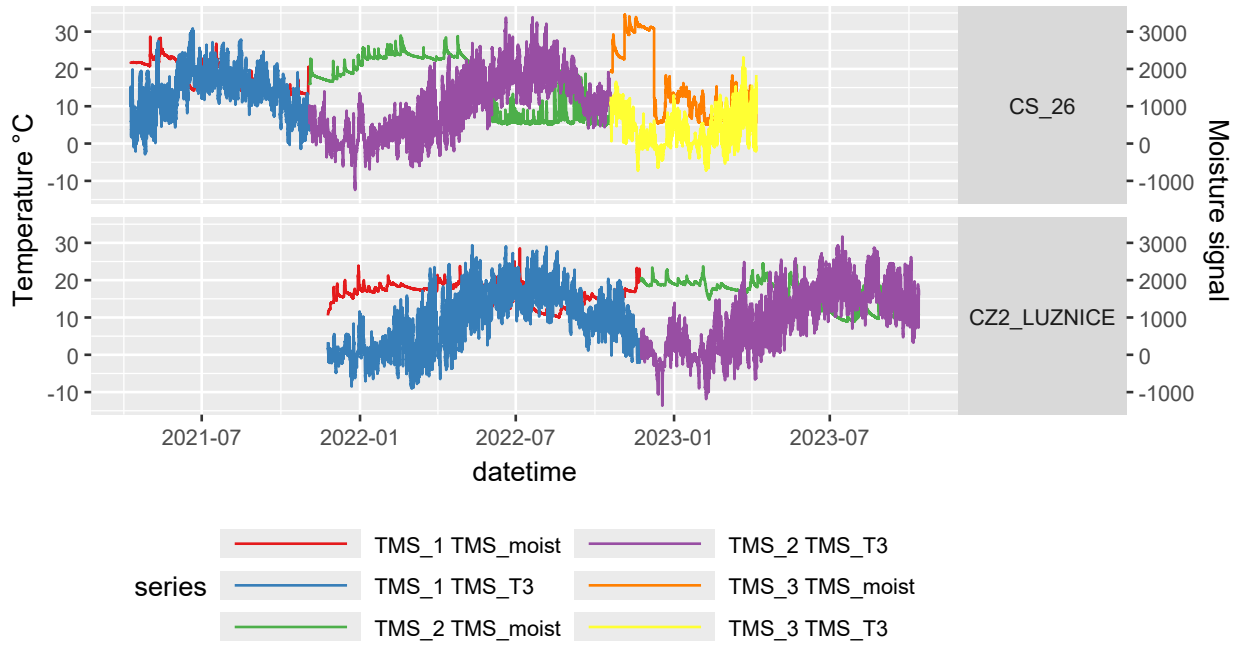

Figure 7: Line graph of temperatures and moisture at two locations.

```
## Raster graph
mc_plot_raster(micro.data.clean[c("CS_26","CZ2_LUZNICE")],
               sensors = "TMS_moist")
```

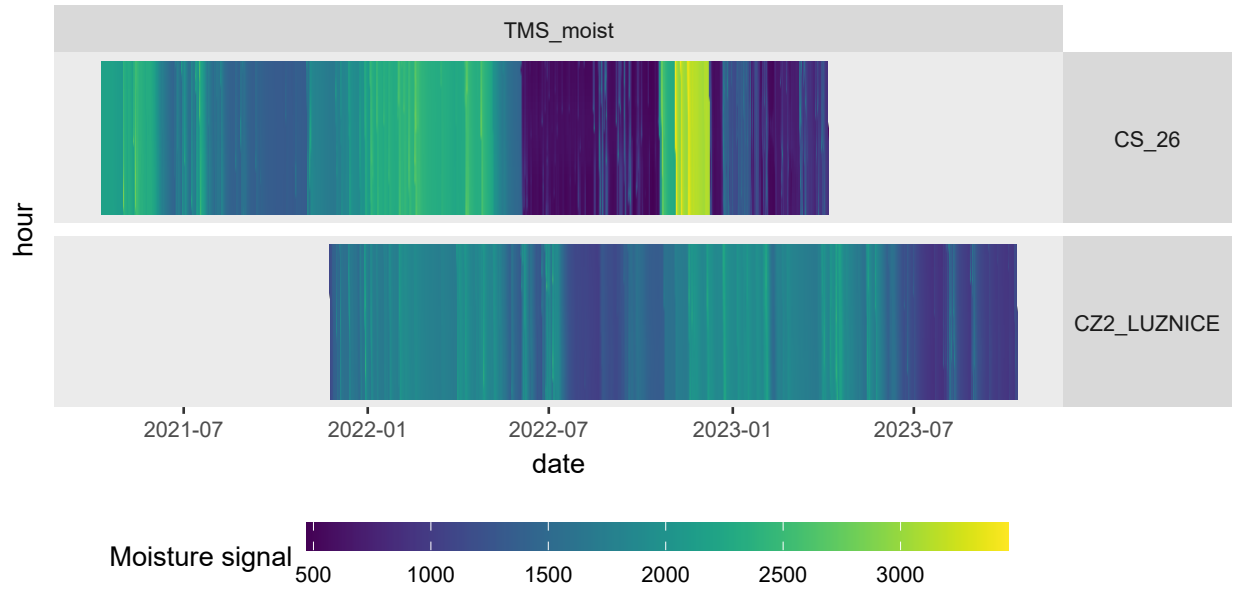

Figure 8: Raster graph of moisture at two locations.

The exact duration of the period when the TMS-4 datalogger was out of contact with soil will help us determine the function `mc_prep_TMSoffsoil` optimized for detection of these events. The function returns a TRUE/FALSE vector, where TRUE means that the datalogger was probably out of soil **(Fig. 9 and 10)**.

```
## Detection of kicked out TMS4 (without soil contact)
micro.data.prep <- mc_prep_TMSoffsoil(micro.data.clean)

## Visualization of detail of location CS_26 using mc_plot_line
# and adjust legend to 3 rows
mc_plot_line(
  micro.data.prep[c("CS_26", "CZ2_LUZNICE")],
  sensors = c("TMS_moist", "off_soil"),
  color_by_logger = TRUE
) + ggplot2::guides(color = ggplot2::guide_legend(nrow = 3))
```

```
## Check for pulling out at all locations
mc_plot_line(micro.data.prep, sensors = "off_soil")
```

From the graph showing soil moisture and the result of detection of irrelevant measurements by the `mc_prep_TMSoffsoil` function, we can see from the combination of colors for individual readings (downloaded files) and sensor TRUE/FALSE = pulled out/in soil that the datalogger was pulled out of soil during June 2022. Subsequently, during the autumn visit to the location in the same year, the datalogger was seated back and in winter 2022/2023 it was pulled out of soil again. Important for us is the first occurrence of TRUE value on the *off_soil* sensor and further information about when it was reinstalled. This information will help us mark and delete compromised measurements from the time when the datalogger was in incorrect position. Jumps between TRUE/FALSE cannot be interpreted as repeated installation attempts and subsequent pulling out, it is an artifact of the pull-out detection method, when the sensor registered soil moisture and temperature values that are within the acceptable range for measurements in soil. We also see problems at location CZ2_CEPICKA, which is confirmed by field notes about moisture sensor failures.

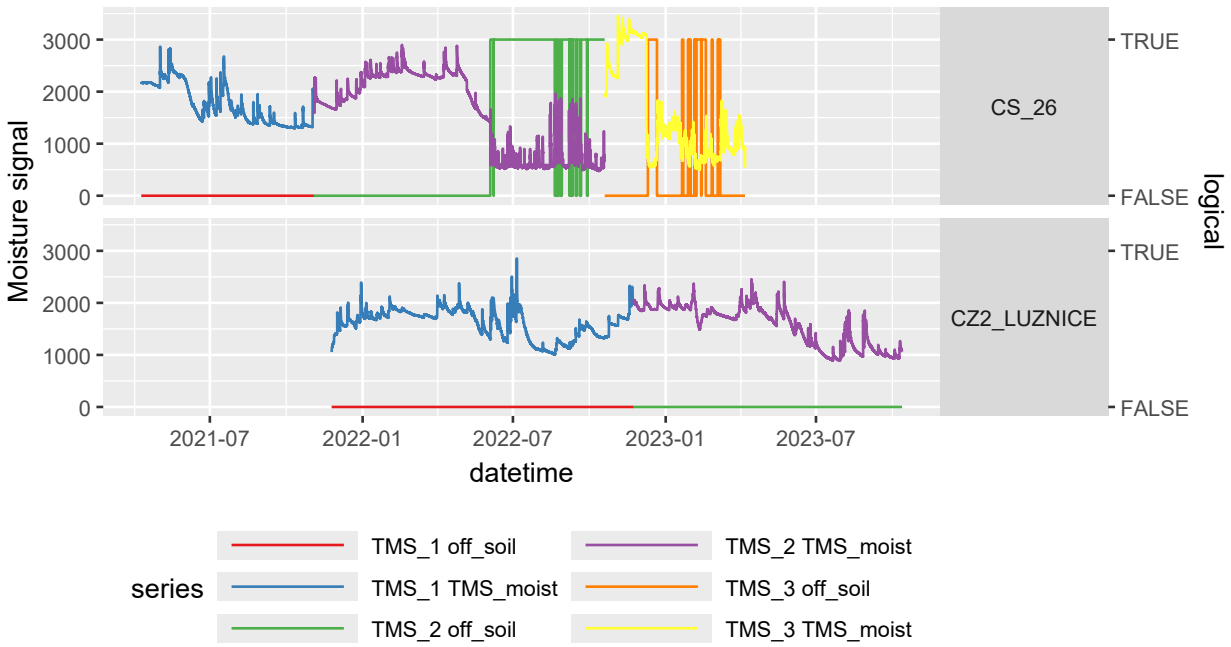

Figure 9: Visualization of logical variable off_soil for one location.

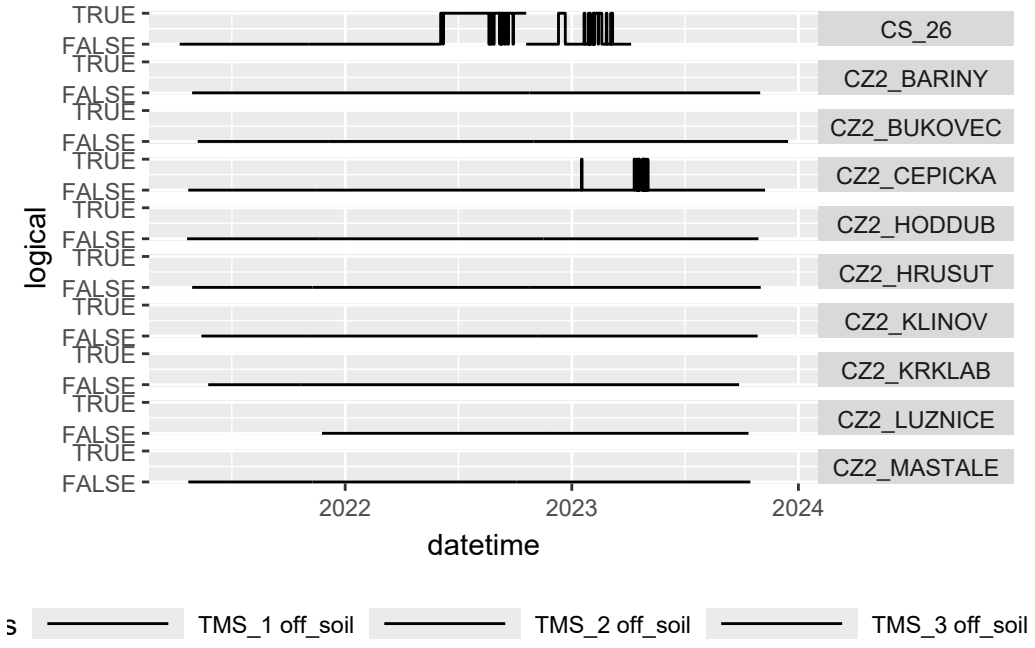

Figure 10: Visualization of logical variable off_soil for multiple locations.

For visual data control, we can use the interactive application *myClimGUI*, which allows loading *myClim* objects, zooming and switching between locations and sensors. Through *myClimGUI* (Fig. 11) it is also possible to mark erroneous/unwanted data for deletion (replacement with NA).

```
## Installation of myClimGUI
remotes::install_github("https://github.com/ibot-geoecology/myClimGui")

## launching application, loading data into myClimGUI
myClimGui::mcg_run(micro.data.prep)
```

Figure 11: Preview of myClimGUI application

It is also possible to mark unwanted measurements using a table where we mark the beginning and end of the unwanted data series. Such an approach has the advantage of better reproducibility than using the interactive *myClimGUI* application, but is less user-friendly. In the interactive application, by zooming on data from location CS_26, we found that the TMS-4 datalogger was pulled out of soil at time 2022-06-03 18:30 and reinstalled at time 2022-10-19 12:00. We will therefore enter these *state* values in the table for all sensors on the given logger where we want to mark data with tag (replace with NA), namely TMS_T1, TMS_T3, TMS_moist. A liberal approach dictates to keep data from sensor TMS_T2, because it is ground temperature and it can be assumed that the pulled out datalogger was lying on the ground. A conservative approach dictates to delete the TMS_T2 time series as well because the integrity of measurement may be compromised.

```
## Getting sample "states/tag" table
states <- mc_info_states(micro.data.prep)

## Saving and editing table in MS excel
write.xlsx(states,"./data/statesR.xlsx")

## Loading edited table with new "error" tags
states.insert <- read.xlsx("./data/states_edit.xlsx")

## Adding new tag value to myClim object
micro.data.prep.s <- mc_states_insert(micro.data.prep,states_table = states.insert)
```

```
## Replacing tag value "error" with NA (delete)
micro.data.prep.na <- mc_states_replace(micro.data.prep.s, tags = "error",
                                        replace_value = NA)

## Removing auxiliary sensor "off_soil"
micro.data.prep.na <- mc_filter(micro.data.prep.na,sensors = "off_soil", reverse = T)
```

Table 4: Marking unusable measurements using tags (from states insert table)

locality_id	logger_name	sensor_name	tag	start	end	value
CS_26	TMS_2	TMS_T1	error	2022-06-03 18:30:00	2022-10-19 12:00:00	offsoil
CS_26	TMS_2	TMS_T3	error	2022-06-03 18:30:00	2022-10-19 12:00:00	offsoil
CS_26	TMS_2	TMS_moist	error	2022-06-03 18:30:00	2022-10-19 12:00:00	offsoil
CS_26	TMS_3	TMS_moist	error	2022-12-08 18:15:00	2023-04-06 13:00:00	offsoil
CS_26	TMS_3	TMS_T1	error	2022-12-08 18:15:00	2023-04-06 13:00:00	offsoil
CS_26	TMS_3	TMS_T3	error	2022-12-08 18:15:00	2023-04-06 13:00:00	offsoil
CZ2_CEPICKA	TMS_1	TMS_T1	error	2022-01-16 23:45:00	2022-01-17 00:45:00	error
CZ2_CEPICKA	TMS_1	TMS_T2	error	2022-01-16 23:45:00	2022-01-17 00:45:00	error
CZ2_CEPICKA	TMS_1	TMS_T3	error	2022-01-16 23:45:00	2022-01-17 00:45:00	error
CZ2_CEPICKA	TMS_2	TMS_moist	error	2022-01-16 23:45:00	2022-01-17 00:45:00	error
CZ2_CEPICKA	TMS_3	TMS_moist	error	2023-01-16 23:45:00	2023-05-05 22:00:00	error

We can check the result of modifications using a graph **(Fig. 12)**.

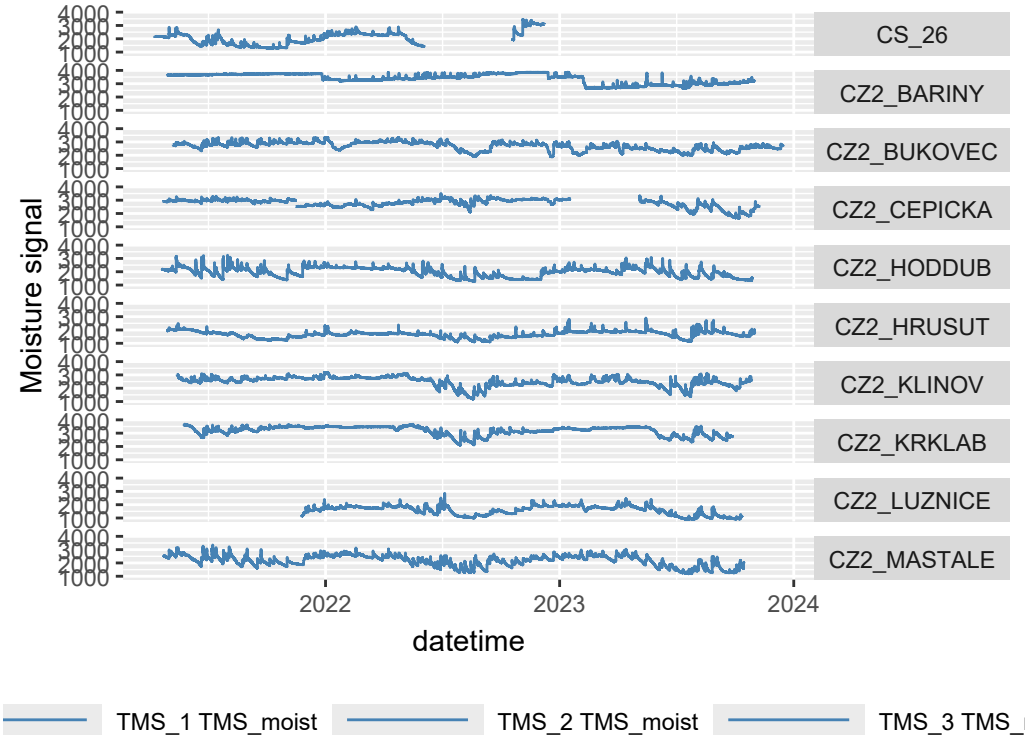

Figure 12: Visual check of soil moisture progression on cleaned time series. Series now do not contain obviously faulty observations.

Temperature Datalogger Calibration

Calibration of datalogger thermometers before/after placement at research plots is desirable. Although the manufacturer states certain measurement accuracy, individual sensors may show larger than manufacturer-stated deviations from actual values due to undesirable external influences or manufacturing defects. Calibration thus helps detect potentially faulty sensors and improve measurement comparability between individual devices. We can perform calibration either in laboratory conditions, or, with lower requirements for measurement accuracy, even by ourselves. For example, by placing a set of microclimatic dataloggers in a location with stable temperature, where we leave them for at least several hours for temperature stabilization (ideally at least 24 hours). From the measured values, we then derive the measurement deviation of individual sensors from the calibrated meter, or from the common average. If only single-point calibration is available, only the correction factor (*cor_factor*) calculated as *reference value - measured value* is applied. If two- or multi-point calibration is available, then the slope of measurement deviation dependency (*cor_slope*) is also applied. The calibrated value is calculated as:

calibrated value = measured value x (cor_slope + 1) + cor_factor

Calibration values for individual sensors, identified by calibration date, datalogger serial number and sensor name, can be assigned to corresponding time series in the *myClim* library in two steps and these can then be cleaned of known measurement error. If a single calibration value is available, it is applied to the entire time series (see example). In case repeatedly determined calibration values are available, they are always applied from the date of calibration with the exception of the first calibration, which is also applied retroactively from the first measurement.

Moisture sensor calibration is covered in chapter 2.3.

```
## Loading table with calibration values (offset)
calib <- read.xlsx("./data/calibrationR.xlsx", detectDates = FALSE)
calib$datetime <- as.POSIXct(convertToDate(calib$datetime))

## Loading calibration data into myClim
micro.data.prep.c <-
  mc_prep_calib_load(micro.data.prep.na, calib_table = calib)

## Application of calibration factor to time series
micro.data.prp.cal <- mc_prep_calib(micro.data.prep.c)
```

Table 5: Calibration data for individual sensors

serial_number	datetime	sensor_id	cor_factor
94171044	2021-10-21	TMS_T1	0.0000
94171044	2021-10-21	TMS_T2	0.0625
94171044	2021-10-21	TMS_T3	0.0625
91201145	2021-10-21	Thermo_T	0.0625
21067844	2021-10-21	HOBO_T	0.0060
21061889	2021-10-28	HOBO_T	0.0110
94181410	2021-10-28	TMS_T1	0.5000
94181410	2021-10-28	TMS_T2	0.1875
94181410	2021-10-28	TMS_T3	0.1250
91201144	2021-10-28	Thermo_T	0.0625

Joining Time Series of Clean Microclimatic Data

We have prepared cleaned and calibrated data and will proceed to joining data series from individual readings to obtain a continuous time series. Joining data using the *myClim* library is an interactive process because

overlaps often occur in the data when two different values are measured at the same moment, for example if dataloggers were changed at the location and data was read from the memory of the new datalogger from the time before measurement started at the location. Rarely, individual duplicate measurements also appear due to negative correction of datalogger internal clock time during reading and subsequent repeated measurement in the measurement interval. It may also be cases of repeated reading of entire datalogger memories, where some segments are thus read multiple times. In such a case, *myClim* automatically preserves only one record per measurement time. In cases where time series overlap but measurement values are not identical, *myClim* prints information about which location, which datalogger and sensor has overlap and what is the time range of overlap and displays an interactive graph zoomed to the overlap location where the user can examine the situation. Then *myClim* waits for user action, who must enter one of the options (digit 1 - 6) into the console according to whether *myClim* should prefer values from the older time series (datalogger) (1), from the newer time series (2), skip this particular overlap and preserve values from both dataloggers (3), for this and all subsequent overlaps in the entire object always prefer older datalogger (4), always prefer newer datalogger (5), or end the joining process (6).

```
# Joining time series
micro.data.join <- mc_join(micro.data.prp.cal)
## visualization of detail of location CS_26
## Due to interactive nature output is not shown here
mc_plot_line(micro.data.join[c("CS_26","CZ2_LUZNICE")],
             sensors = c("TMS_moist","TMS_T1"),
             color_by_logger = TRUE)
```

The output of joining can be checked visually in *myClimGUI*, for example the graph of locations 'CS_26' and 'CZ2_LUZNICE' is displayed again with parameter `color_by_logger = TRUE` similar to above. The graph shows that we have removed problematic measurements and joined the time series into one whole.

Filling Missing Measurements

The last step in preparing microclimatic data for analysis is replacing missing values. Using *myClim*, it is safe to fill especially short dropouts in the order of 1 - 5 missing observations depending on the time step of the given datalogger, for example when data for individual readings do not exactly connect. Such values can be filled using simple linear interpolation of two neighboring measurements implemented in *myClim*. We do not recommend filling longer segments using *myClim*.

```
## Filling missing values with maximum length of 10 consecutive measurements
micro.data.fill <- mc_prep_fillNA(micro.data.join,maxgap = 10)
```

Saving Data Prepared for Analysis

If when loading microclimatic data from files we did not use the option to load supplementary information about locations, it is possible to add it additionally. In this example, we will add information about geographical coordinates of locations. These will come in handy during further processing, for example for calculating local time shift relative to coordinated universal time (UTC) for synchronizing photoperiod across locations. Finally, we save the object containing all time series of measurements and metadata about locations to a *.rds* file. A *myClim* object saved in a *.rds* file is data efficient (in this case, the file 'prep_CZ2_10_join.rds' takes up only 11 MB compared to 129 MB of source .csv files). For saving a *myClim* object, we always recommend using the function *myClim::mc_save()*, which ensures backward compatibility of the object with new versions of the *myClim* library, instead of the basic function *saveRDS()*.

```r
## Add longitude in WGS84 coordinate system (lon_wgs84)
# in decimal degrees
micro.data.fill <- mc_prep_meta_locality(micro.data.fill,
                                         list(CZ2_KRKLAB = 15.5640182,
                                              CZ2_KLINOV = 12.9735495,
                                              CZ2_BARINY = 17.9556943,
                                              CZ2_HRUSUT = 17.4393209,
                                              CZ2_MASTALE = 16.1369899,
                                              CZ2_HODDUB = 17.0748401,
                                              CZ2_CEPICKA = 16.3635405,
                                              CZ2_LUZNICE = 14.8428597,
                                              CS_26 = 13.9910361,
                                              CZ2_BUKOVEC = 15.3606035),
                                         param_name = "lon_wgs84")
## Add latitude in WGS84 coordinate system (lat_wgs84)
# in decimal degrees
micro.data.fill <- mc_prep_meta_locality(micro.data.fill,
                                         list(CZ2_KRKLAB = 50.7512076,
                                              CZ2_KLINOV = 50.3935486,
                                              CZ2_BARINY = 49.6310008,
                                              CZ2_HRUSUT = 49.6932542,
                                              CZ2_MASTALE = 49.8115909,
                                              CZ2_HODDUB = 48.8803506,
                                              CZ2_CEPICKA = 49.4982277,
                                              CZ2_LUZNICE = 48.9940496,
                                              CS_26 = 50.5953875,
                                              CZ2_BUKOVEC = 50.8143946),
                                         param_name = "lat_wgs84")
## Saving final myClim object
mc_save(micro.data.fill, file = "./data/prep_CZ2_10_join.rds")
```

2. Microclimatic Variables

In previous chapters, we focused on how to create the most relevant microclimatic measurements from dataloggers, how to download them from dataloggers, clean them, validate them, calibrate them, and possibly connect time series from one location with multiple readings.

In this section, we will show how to derive biologically relevant microclimatic variables from time series of microclimate measurements. These characterize the microclimate at locations over a given time period (e.g., annual/seasonal/monthly/daily averages, minima, maxima, or variability).

2.1. Time / Period

Microclimatic dataloggers typically measure at relatively fine temporal resolution (tens of seconds to hours). However, for biological analyses, it is usually desirable to aggregate data over time using mathematical functions and thus obtain descriptive characteristics for a given period (e.g., daily, monthly, seasonal, annual...). The choice of period length for calculating microclimatic variables depends on the question we want to answer using microclimatic data. For some systems and variables, it is appropriate to aggregate time series over the **calendar year**. Where soil moisture influenced by snow melt comes into consideration, the **hydrological year** is a more suitable choice (for the Czech Republic, the period November 1 - October 31). If part of the year is not relevant for the studied system, then it may be desirable to use microclimatic data from a specific period, e.g., **growing season**. The choice of appropriate time period for data aggregation plays an important role because it affects whether we capture biologically relevant signal in the data or whether it is obscured by noise from measurements in a period that is not relevant for the given question.

For processing microclimatic time series, the time zone in which loggers measure is also important, especially when analyzing data from a larger area, e.g., globally. Time shift can play a role especially in data aggregations and calculations of biologically relevant variables. Many loggers measure in UTC (TOMST TMS-4) regardless of user preference or time zone where they are installed, other loggers can be set to a specific time zone. The *myClim* library used here strictly assumes that the time series of any input data is in UTC. If a user uses loggers that don't measure in UTC, it is desirable to specify the `tz_offset` parameter when loading them into *myClim*, i.e., the number of minutes by which the given time series is shifted relative to UTC. *myClim* will then take this offset into account in subsequent aggregations.

For examples, we use data from ten locations. All locations have TOMST TMS-4 loggers measuring air temperature at 15 cm, 2 cm above soil surface, 8 cm below soil surface, soil moisture in the top layer 0 - 15 cm, nine locations have Thermologger measuring air temperature at 200 cm above ground, and HOBO U23 logger measuring air temperature and humidity at 150 cm above ground. Time series approximately from April 2021 to October 2023, but loggers did not measure for the same duration, there are missing segments in the data.

```
## Loading prepared myClim object from chapter 1 - Data Preparation from Files
# Installation and loading of libraries
suppressWarnings(lapply(c("myClim", "ggplot2"), function(pkg) {
  if (!require(pkg, character.only = TRUE))
    install.packages(pkg, repos = "https://mirrors.nic.cz/R/")
  library(pkg, character.only = TRUE)
}))

micro.data <- mc_load("./data/prep_CZ2_10_join.rds")
micro.data
```

```
## plotting graph of temperature and relative humidity progression
p <- mc_plot_line(micro.data[5:7], sensors = c("TMS_T3", "HOBO_RH"))
```

```r
## manual color adjustment
p <-
  p + ggplot2::scale_color_manual(values = scales::alpha(c("darkblue", "hotpink"), 0.7),
                                  name = NULL)
print(p)
```

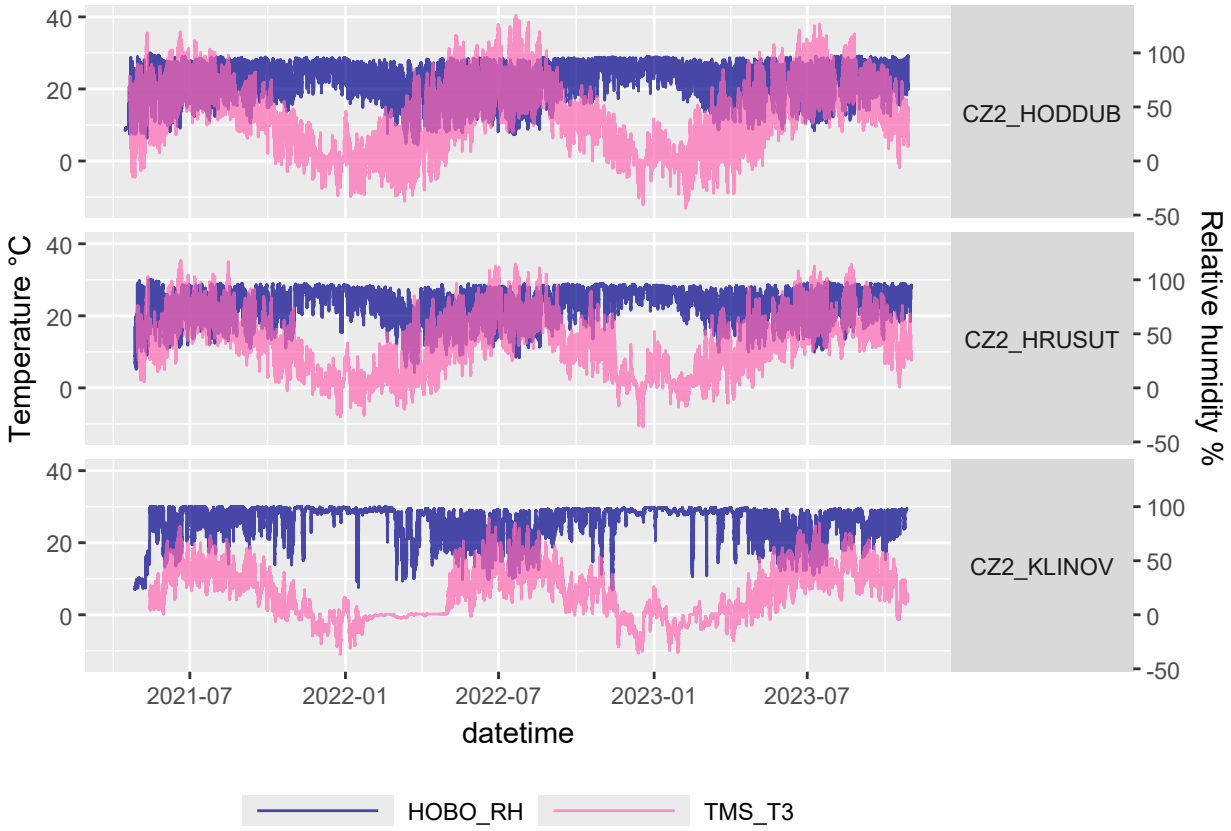

Figure 13: Plotting graph of temperature and relative humidity progression.

```r
## calculation of offset relative to UTC, photoperiod synchronization
micro.data<-mc_prep_solar_tz(micro.data)
```

Table 6: Display of information about micro.data object

locality_id	lon_wgs84	lat_wgs84	elevation	tz_offset
CS_26	13.99104	50.59539	NA	56
CZ2_BARINY	17.95569	49.63100	NA	72
CZ2_BUKOVEC	15.36060	50.81439	NA	61
CZ2_CEPICKA	16.36354	49.49823	NA	65
CZ2_HODDUB	17.07484	48.88035	NA	68
CZ2_HRUSUT	17.43932	49.69325	NA	70
CZ2_KLINOV	12.97355	50.39355	NA	52
CZ2_KRKLAB	15.56402	50.75121	NA	62
CZ2_LUZNICE	14.84286	48.99405	NA	59
CZ2_MASTALE	16.13699	49.81159	NA	65

2.2. Temperature

Temperature and variables directly derived from it are among the most used (micro)climatic parameters. This is partly due to actual biological relevance, but also due to the low technical demands of temperature measurement. Basic temperature variables undoubtedly include average temperature for a given period and temperature extremes. Average is a methodologically easily graspable variable, however, even here we encounter different calculation methods. The meteorological dictionary (https://slovnik.cmes.cz/) refers to the arithmetic average of all regular observations during 24 hours as "true daily average temperature" and this calculation method is preferred in micrometeorology. In meteorology in the Czech Republic, for historical reasons, daily average air temperature is used as a weighted average of temperatures at measurement times 7h, 14h and 21h local time, with measurement at 21h having double weight. American NOAA defines daily average temperature as the average of minimum and maximum daily temperature, which can lead to certain deviations in temperatures thus determined and complicate comparability of reported values from different sources.

Similarly, the method of determining minimum and maximum temperatures differs in practice. In SYNOP reports, the minimum from temperatures during the night period (from 18 to 06 UTC) is reported, for climatological purposes minimum temperature is reported for the period 24 hours before the evening term (21h). To eliminate the influence of outlier observations (due to random climatic events or influenced measurements), percentiles (e.g., 5th percentile for minima and 95th percentile for maxima) from observed values are used in microclimatology to characterize temperature extremes.

An important derived variable is then the sum of temperatures above or below a certain threshold temperature (sum of effective temperatures, Growing degree days, GDD / Freezing degree days, FDD). These determine, for example, the length of season for crop or pest development, or conversely the length of freezing relevant for survival of some organisms (pests / parasites). The choice of threshold temperature depends on the research question; in agronomy, different threshold values are established for individual crops according to their physiological requirements. In ecology, a threshold value of 5°C is generally used for calculation, which is widely accepted as the limiting temperature for plant growth.

Calculation of Virtual Sensors from Temperature

For easier work with data, it is useful to store some variables, derived from one or more sensors, directly as a so-called virtual sensor of the given logger. These sensors have the same measurement period as those sensors from which they were derived. For example, from temperatures we calculate virtual sensor Growing degree days (GDD) or Freezing degree days (FDD), i.e., the sum of effective/frost temperatures above/below chosen threshold value - the sensor stores values by which each period contributes to the given variable. Only after aggregation to days or longer time period can we speak directly about FDD or GDD.

```
# Instalation and loading of libraries
suppressWarnings(lapply(c("myClim", "dplyr", "tidyr"), function(pkg) {
  if (!require(pkg, character.only = TRUE))
    install.packages(pkg, repos = "https://mirrors.nic.cz/R/")
  library(pkg, character.only = TRUE)
}))
# For example we'll use TMS_T3 sensor 15 cm above ground, from TMS-4 TOMST logger
micro.data <- mc_calc_gdd(micro.data, sensor = "TMS_T3", t_base = 5)
micro.data <- mc_calc_fdd(micro.data, sensor = "TMS_T3", t_base = 5)
```

Calculation of Temperature Variables

For example, let's define growing season as the period May 1 - August 31. Using parameter `period="custom"` we can introduce any period for which variables will be calculated. If the time series contains measurements

from multiple years, variables are calculated for all years where there is sufficient data coverage defined by parameter `min_coverage=0.9`, i.e., if we have more than 90% of data for the given period, the variable is calculated, if less than 90% myClim returns NA, i.e., empty value. Aggregation functions are defined in the example below as a list of functions, where we specifically select which functions apply to which sensors. Sensors for which no function is defined do not enter the calculation and are not included in the resulting object. Parameter `percentiles = c(5,95)` defines which percentiles should be calculated in the "percentile" function.

```
## display sensors available in myClim object for calculations
levels(factor(mc_info(micro.data)$sensor_name))
```

```
## [1] "FDD5"      "GDD5"      "HOBO_RH"   "HOBO_T"    "Thermo_T"  "TMS_moist"
## [7] "TMS_T1"    "TMS_T2"    "TMS_T3"
```

```
## select temperature sensors and required aggregation functions
micro.veget <- mc_agg(micro.data,
                      period = "custom",
                      custom_start = "05-01",
                      custom_end = "08-31",
                      percentiles = c(5,95),
                      min_coverage = 0.9,
                      fun=list(
                        FDD0="sum",
                        GDD5="sum",
                        HOBO_T=c("mean","percentile","range"),
                        Thermo_T=c("mean","percentile","range"),
                        TMS_T1=c("mean","percentile","range"),
                        TMS_T2=c("mean","percentile","range"),
                        TMS_T3=c("mean","percentile","range")))
```

The resulting myClim object contains all input sensors with suffix of given function. Each sensor has number of measurements according to how many segments of time series met the condition "min_coverage". In our example these are for some locations 3 values (3 growing seasons), in some cases 2 values, sometimes only one because input time series was too short.

```
## display resulting sensors
levels(factor(mc_info(micro.veget)$sensor_name))
```

```
##  [1] "GDD5_sum"              "HOBO_T_mean"           "HOBO_T_percentile5"
##  [4] "HOBO_T_percentile95"   "HOBO_T_range"          "Thermo_T_mean"
##  [7] "Thermo_T_percentile5"  "Thermo_T_percentile95" "Thermo_T_range"
## [10] "TMS_T1_mean"           "TMS_T1_percentile5"    "TMS_T1_percentile95"
## [13] "TMS_T1_range"          "TMS_T2_mean"           "TMS_T2_percentile5"
## [16] "TMS_T2_percentile95"   "TMS_T2_range"          "TMS_T3_mean"
## [19] "TMS_T3_percentile5"    "TMS_T3_percentile95"   "TMS_T3_range"
```

```
## convert from myClim object to simple table, long format
df.veget <- mc_reshape_long(micro.veget)

## visualization
## average, minimum and maximum air temperature 200 cm above ground (Thermo_T)
df.veget$datetime <- as.character(df.veget$datetime)
```

```r
viz <- df.veget %>%
  filter (
    sensor_name %in% c(
      "Thermo_T_mean",
      "Thermo_T_percentile5",
      "Thermo_T_percentile95"

    )
  ) %>%
  filter (!is.na(value))

viz <- pivot_wider(viz,
                   names_from = "sensor_name",
                   values_from = "value")

p <-
  ggplot(viz, aes(x = locality_id, y = Thermo_T_mean, col = datetime)) +
#  ggplot(viz, aes(x = locality_id, y = TMS_T1_mean, col = datetime)) +
  geom_point(position = position_dodge(width = 0.5)) +
  geom_errorbar(
    aes(ymin = Thermo_T_percentile5,
        ymax = Thermo_T_percentile95,
#    aes(ymin = TMS_T1_percentile5,
#        ymax = TMS_T1_percentile95,
        width = 0.15),
    position = position_dodge(width = 0.5)
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(
    angle = 60,
    vjust = 0.7,
    hjust = 0.5
  )) +
  theme(legend.position = "bottom") +
  labs(title = "Temperature at 2 m",
       subtitle = "average, minimum (P5) and maximum (P95)")

print(p)
```

Calculation of myClim Temperature Variables

For calculating statistics aggregating measurements over longer time periods, the `mc_agg()` function serves in the *myClim* library. It applies functions (predefined or user-defined) to measurement series in a specified time period.

For calculating a standardized set of temperature indicators, the `mc_env_temp()` function serves in the *myClim* library, which calculates the following summary indicators for a specified time period:

- **min5p**: Minimum temperature (5th percentile of daily minimum temperatures)
- **mean**: Average temperature (average of daily temperatures)
- **max95p**: Maximum temperature (95th percentile of daily maxima)
- **drange**: Average daily temperature amplitude (average difference between daily maxima and minima)

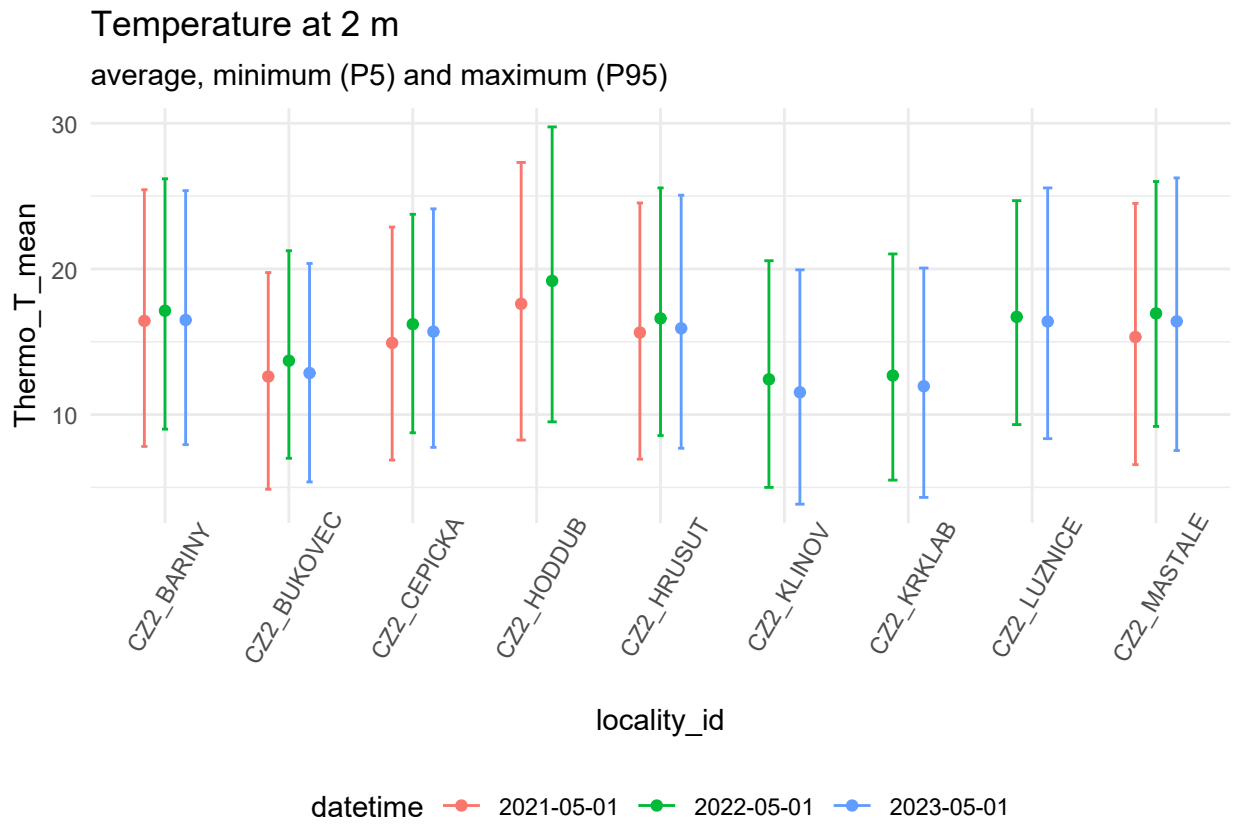

Figure 14: Average, minimum and maximum temperature at 2 m at several locations.

- **GDD5**: Sum of effective temperatures (sum of temperatures above threshold value, default value 5°C)
- **FDD0**: Sum of frost temperatures (sum of negative temperature deviations from threshold value, default value 0°C)
- **frostdays**: Number of frost days (days with minimum temperature <0°C)

When manually calculating using the `mc_agg()` function, values are directly aggregated according to user-specified parameters. In contrast, when using the `mc_env_temp()` function, temperature variables are first aggregated to daily step and from there then to the period specified by the user. This approach is closer to meteorological standards. Another difference is that the `mc_env_temp()` function returns a flat table directly. This cannot be further processed using the *myClim* library, but is directly prepared for subsequent analyses.

The `mc_env_temp()` function requires that all sensors at locations have the same time step. In our data set that we use as an example, this is not the case. There are 2 types of loggers at the locations: TOMST TMS measures at 15-minute intervals and HOBO at 30-minute intervals. The simplest solution is to unify the time step to 30 minutes using aggregation with, for example, average `mc_agg(period="30 min", fun="mean")`.

For example, here we will calculate variables for the calendar year. Here too we define minimum data coverage, or maximum allowed proportion of missing values using the `min_coverage` parameter. In the resulting table there is a large number of rows with missing value, which indicates that the condition `min_coverage` was not met for the given period. We can consider different threshold value setting, missing (NA) values can be subsequently removed from the resulting table.

```
## unify period of TMS and HOBO loggers to 30 min
micro.data30 <- mc_agg(micro.data, period = "30 min", fun = "mean")

## calculation of myClim temperature variables for growing season
micro.temp <-
  mc_env_temp(
    micro.data30,
    period = "year",
    min_coverage = 0.9,
    gdd_t_base = 5,
    fdd_t_base = 0
  )

## remove missing values
micro.temp <- filter(micro.temp, !is.na(value))

## visualization
micro.temp$fun <- micro.temp$sensor_name %>%
  strsplit(".", fixed = T) %>%
  lapply("[", 3) %>%
  unlist()

viz <- micro.temp %>%
  filter (fun %in% c("mean", "min5p", "max95p")) %>%
  filter (!is.na(value))
viz$sensor_name <- NULL
viz <- pivot_wider(viz, names_from = "fun",
                   values_from = "value")

p <- ggplot(viz, aes(x = locality_id, y = mean, col = height)) +
  geom_point(position = position_dodge(width = 0.5)) +
  geom_errorbar(aes(ymin = min5p,
```

```
                    ymax = max95p,
                    width = 0.15),
                position = position_dodge(width = 0.5)) +
  theme_minimal() +
  theme(axis.text.x = element_text(
    angle = 60,
    vjust = 0.7,
    hjust = 0.5
  )) +
  theme(legend.position = "bottom") +
  ylab("Temperature [°C]") +
  xlab(NULL) +
  labs(title = "Temperature at different heights/depths",
       subtitle = "average, minimum (P5) and maximum (P95) in year 2022")

print(p)
```

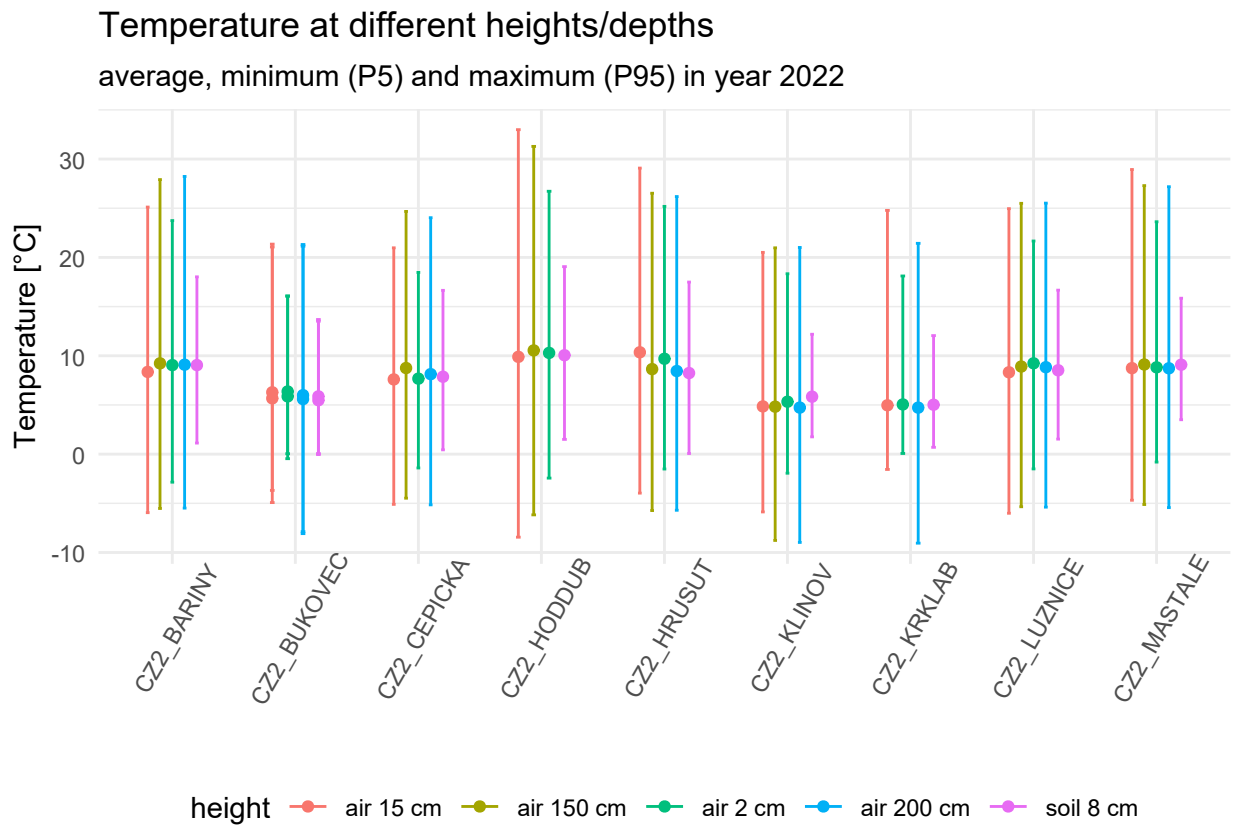

Figure 15: Average, minimum and maximum temperature at different heights at several locations.

2.3. Soil Moisture

The numerical output of soil moisture sensors in microclimatic loggers often is not directly volumetric soil moisture values or other standard variables. Raw moisture signal values from loggers can be used directly for relative comparison or for obtaining rough information about moisture conditions in the given system. However, it is more appropriate to try to convert raw measurements to standard quantities if the logger manufacturer provides necessary information and calibration protocols.

Calculation of Volumetric Soil Moisture

Soil moisture sensors (TOMST TMS4) provide information about volumetric soil moisture after processing the raw signal. This quantity gives a basic idea about moisture availability for plants and other processes dependent on soil water content. However, hydrolimits for individual soil types differ, and therefore measured values need to be interpreted in relation to soil properties at the specific site. For example, the wilting point ranges between 5% volumetric moisture for sandy soils to 20% for clay soils.

Converting TMS4 raw moisture signal to volumetric moisture is still subject to discussion and optimization. There is a considerable range of options for which soil type to use whether by selection from pre-defined ones or by entering calibration parameters of own soil type. Logger calibration also plays a role (values that the sensor reaches in air and after immersion in water). For more information we recommend the following sources:

- help for function mc_calc_vwc()
- calibration application on manufacturer's website TMS Calibr utility

For simplification here we will use the universal calibration curve derived in work (Kopecký et al., 2021). Besides conversion to volumetric moisture, the `mc_calc_vwc()` function in basic setting also deletes moisture values in period when soil was frozen `frozen2NA = TRUE`, because such measurements are biologically less relevant.

```
## calculation of virtual sensor for volumetric moisture
## conversion of TMS moisture signal to volumetric moisture
moist.data <- mc_calc_vwc(micro.data, soiltype = "universal",
                          frozen2NA = TRUE)

## visualization
p <- mc_plot_line(moist.data,sensors = c("TMS_moist", "VWC_moisture"))
p <- p + ggplot2::scale_color_manual(values = c("darkblue","steelblue"),
                                     name = NULL)
print(p)
```

Soil Moisture Variables

For calculating standardized soil moisture indicators, the `mc_env_moist()` function serves in the *myClim* library, which calculates the following parameters for time series of moisture measurements for specified time intervals:

- **VWC.5p**: minimum volumetric soil moisture (5th percentile of volumetric soil moisture)
- **VWC.mean**: average volumetric soil moisture
- **VWC.95p**: maximum volumetric soil moisture (95th percentile of volumetric soil moisture)
- **VWC.sd**: stability of water regime (standard deviation of volumetric soil moisture)

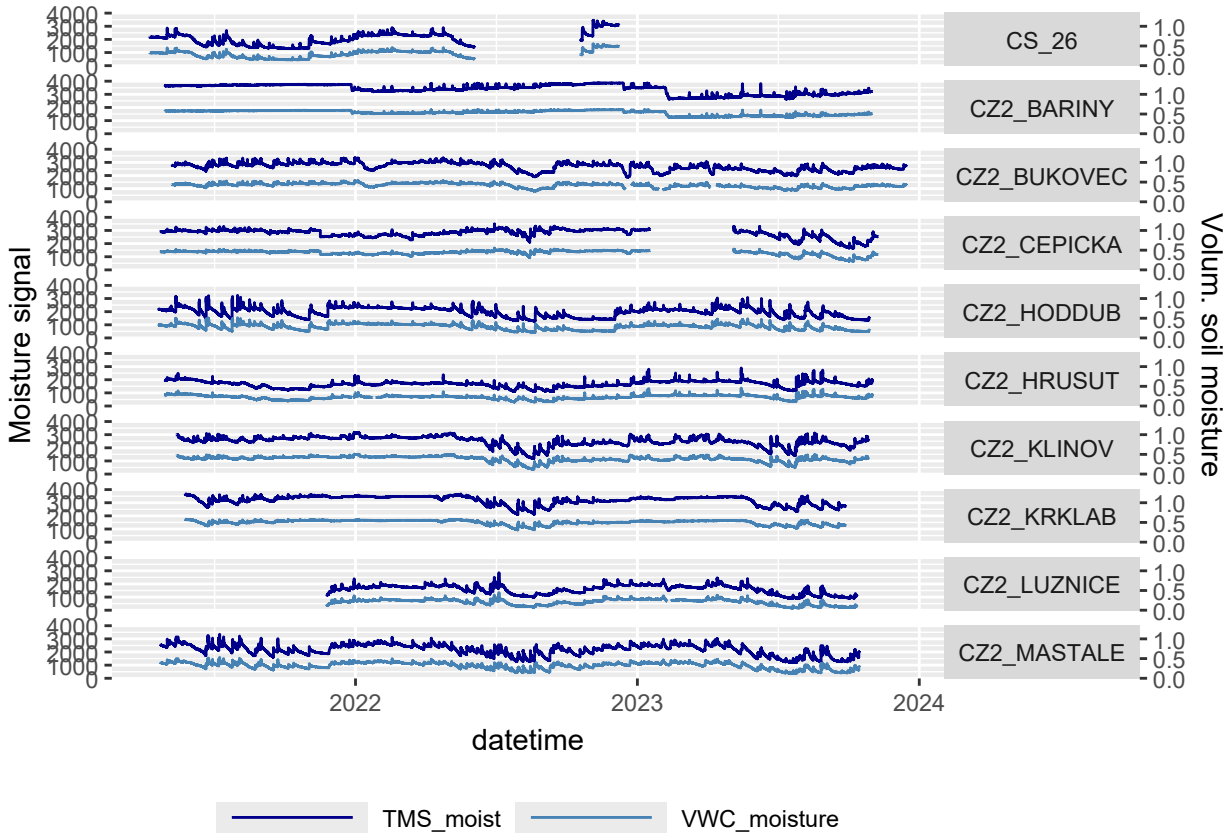

Figure 16: Visualization of raw moisture signals and volumetric moisture progression for several locations.

Similar to the function for calculating *myClim* temperature variables, here too the calculated variables are returned in the form of a flat table that can no longer be used in the *myClim* library but is prepared for subsequent analyses. Before using the `mc_env_moist()` function, it is necessary to first calculate virtual sensors of volumetric moisture. The function does not work with TMS units or with other direct measurements that are not volumetric moisture.

Here too the function requires that all sensors at locations have the same time step. In our data set that we use as an example, this is not the case. There are 2 types of loggers at the locations: TOMST TMS measures at 15-minute intervals and HOBO at 30-minute intervals. We can similarly as in the temperature example above convert sensors to the same step. In this case however we will remove HOBO loggers from the myClim object because we don't need them for this calculation. Here too the function returns us a large number of missing values due to the `min_coverage` parameter.

```
## use myClim object with virtual sensor for volumetric moisture
## remove HOBO loggers from myClim object, they have different time step
moist.data.TMS <-
  mc_filter(moist.data, logger_types = "HOBO_U23-001A", reverse = T)

## calculation of variables only for TOMST TMS loggers
micro.moist <-
  mc_env_moist(moist.data.TMS, period = "year", min_coverage = 0.9)

## remove missing values
micro.moist <- filter(micro.moist, !is.na(value))

## visualization
micro.moist$fun <- micro.moist$sensor_name %>%
  strsplit(".", fixed = T) %>%
  lapply("[", 3) %>%
  unlist()
micro.moist$sensor_name <- NULL
micro.moist.w <- pivot_wider(micro.moist, names_from = "fun",
                             values_from = "value") %>%
  filter(datetime == as.Date("2022-01-01"))

ggplot(micro.moist.w, aes(x = locality_id, y = mean)) +
  geom_point() +
  geom_errorbar(aes(ymin = `5p`,
                    ymax = `95p`,
                    width = 0.15)) +
  theme_minimal() +
  theme(axis.text.x = element_text(
    angle = 60,
    vjust = 0.7,
    hjust = 0.5
  )) +
  theme(legend.position = "bottom") +
  ylab("Volumetric soil moisture [%]") +
  xlab(NULL) +
  labs(title = "Volumetric soil moisture",
       subtitle = "average, minimum (P5) and maximum (P95) for year 2022")
```

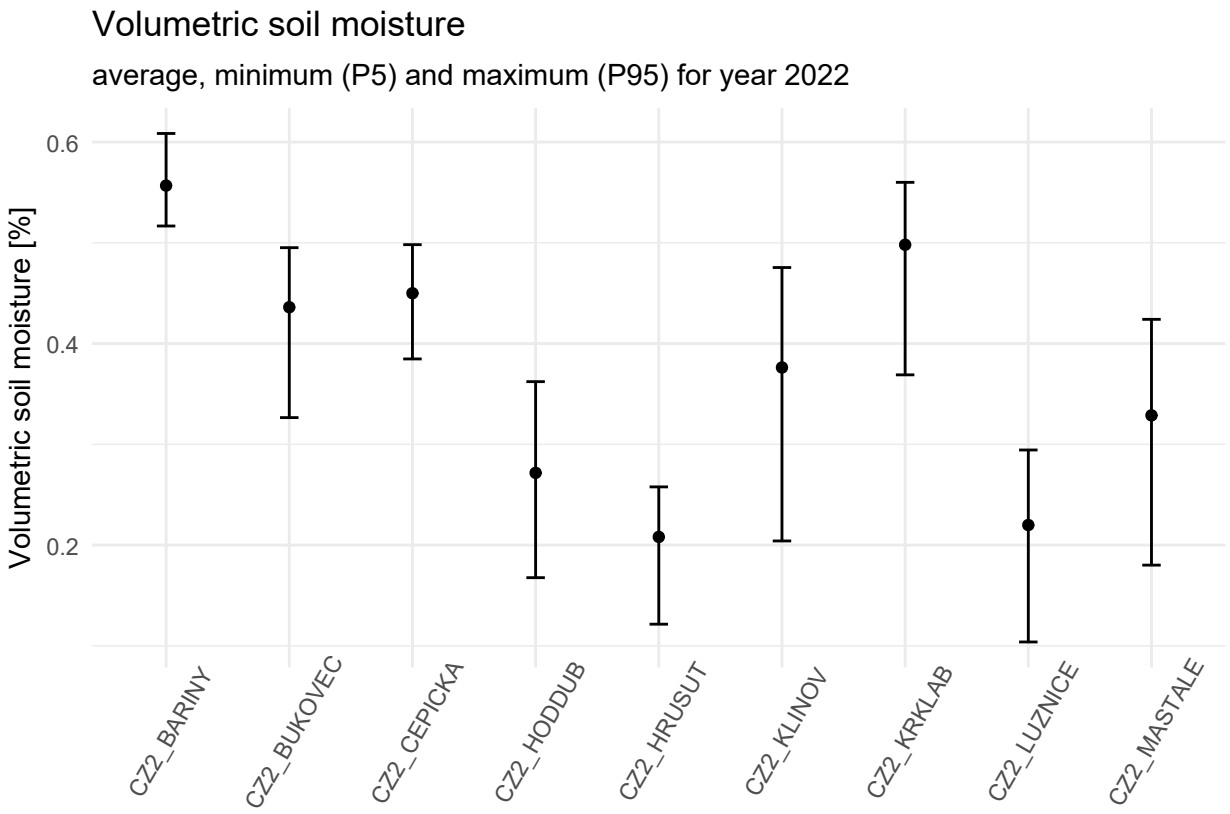

Figure 17: Volumetric soil moisture at several locations.

2.4. Air Humidity

From the perspective of biological relevance, vapor pressure deficit (VPD) is an important indicator of air humidity. It expresses the difference between maximum water vapor pressure at given temperature and actual water vapor pressure and thus reflects, better than relative air humidity, e.g., water loss by transpiration and risk of water stress in plants or fire risk. With knowledge of temperature and relative air humidity (e.g., from HOBO U23 Pro v2 sensor data), vapor pressure deficit can be calculated using the `mc_calc_vpd()` function.

The function also requires that all sensors at locations have the same time step. Again, we will remove loggers from the myClim object that we won't need for this calculation, we will keep only HOBO loggers. Here too the function returns us a large number of missing values due to the `min_coverage` parameter.

```
## keep only HOBO loggers
micro.data.HOBO <- mc_filter(micro.data,logger_types = "HOBO_U23-001A",reverse = F)

## calculation of virtual sensor for vapor pressure deficit (VPD) from HOBO loggers
vpd.data <- mc_calc_vpd(micro.data.HOBO)

## visualization
p <- mc_plot_line(vpd.data,sensors = c("HOBO_RH", "VPD"),scale_coeff = 20)
p <- p + ggplot2::scale_color_manual(values = c("red","black"),
                                     name = NULL)
print(p)
```

myClim Air Humidity Variables

For calculating standardized air humidity variables (vapor pressure deficit VPD), the `mc_env_vpd()` function serves in the *myClim* library, which calculates the following parameters for time series of virtual sensor VPD:

- **VPD.mean**: average vapor pressure deficit (average of daily VPD averages)
- **VPD.max95p**: maximum vapor pressure deficit (95th percentile of daily VPD maxima)

```
## use myClim object with virtual VPD sensor calculated above
micro.vpd <- mc_env_vpd(vpd.data, period = "year", min_coverage = 0.9)

## remove missing values
micro.vpd <- filter(micro.vpd, !is.na(value))

## visualization
micro.vpd$proměnná <- micro.vpd$sensor_name %>%
  strsplit(".", fixed = T) %>%
  lapply("[", 3) %>%
  unlist()

ggplot(micro.vpd, aes(x = locality_id, y = value, col = proměnná)) +
  geom_point() +
  theme_minimal() +
  theme(legend.position = "bottom") +
  ylab("Vapor pressure deficit [kPa]") +
  xlab(NULL) +
  labs(title = "Vapor pressure deficit",
       subtitle = "average and maximum (P95) for year 2022")
```

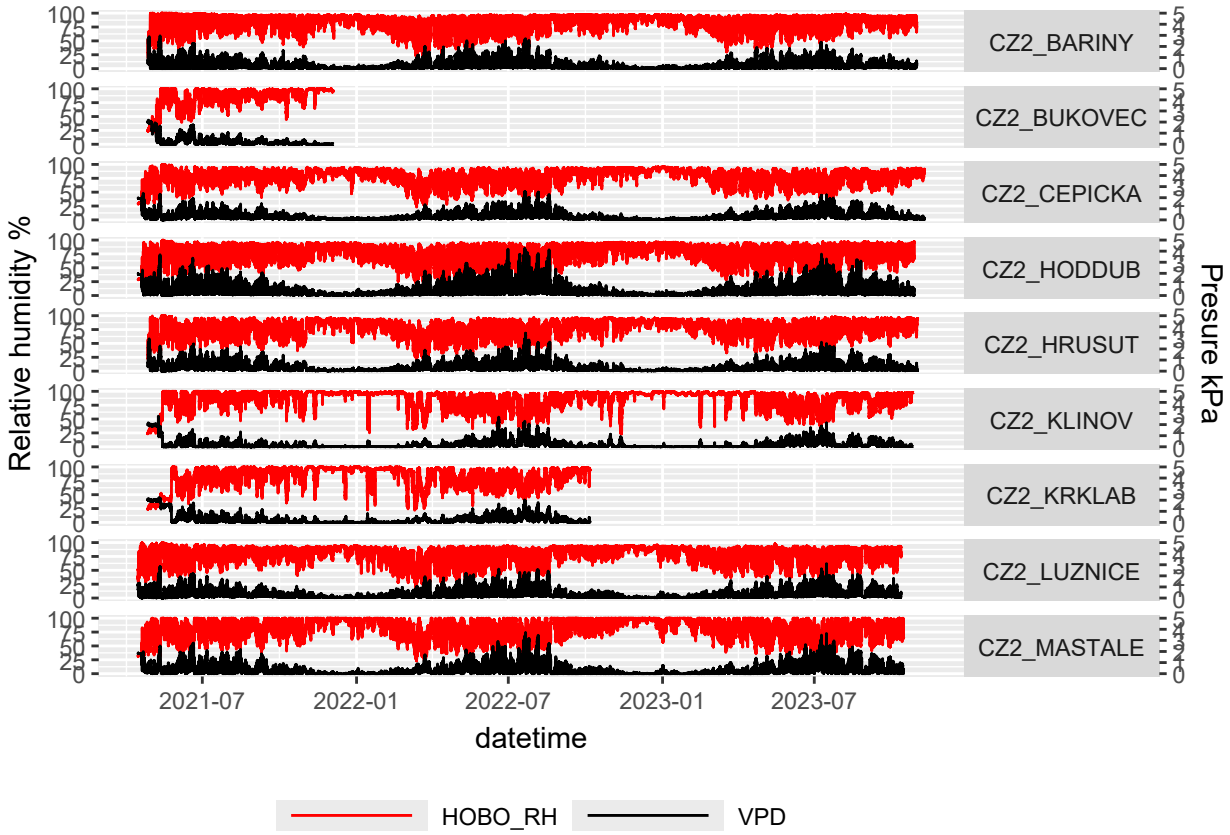

Figure 18: Relative humidity and vapor pressure deficit progression for several locations.

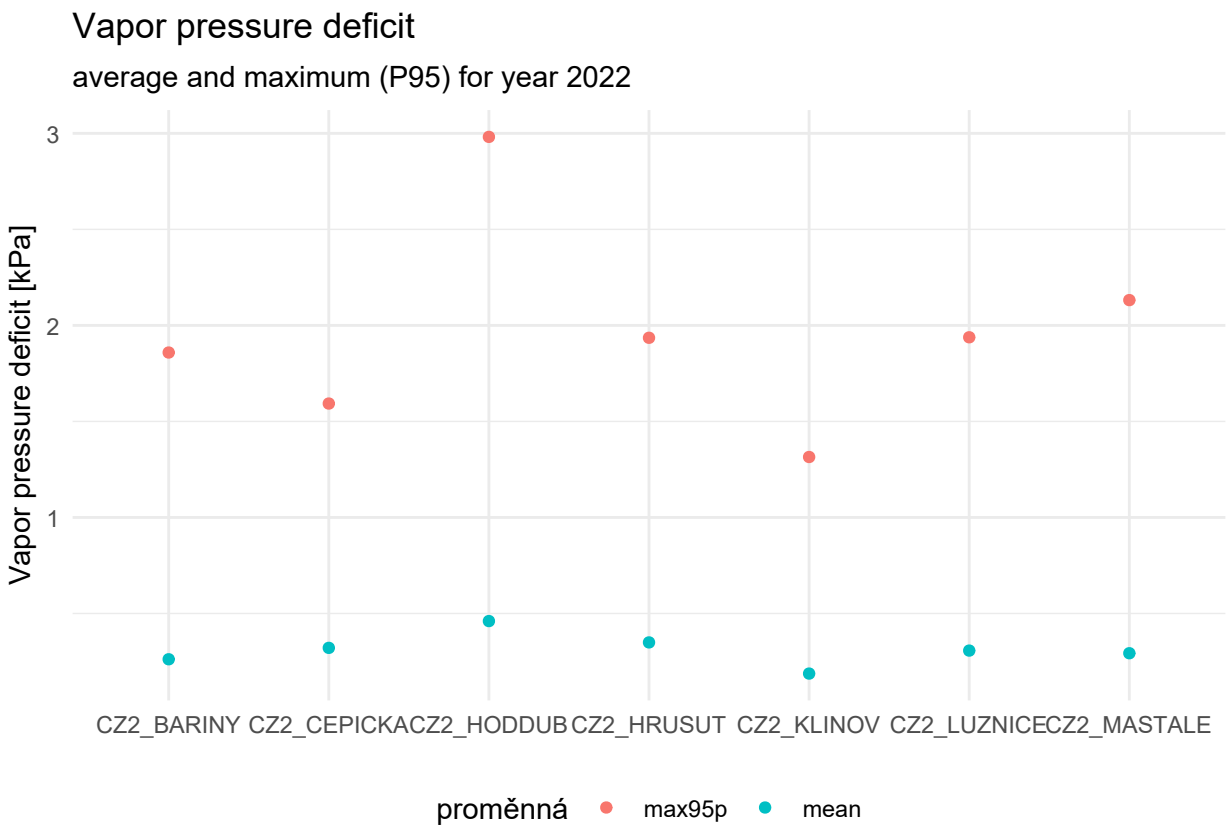

Figure 19: Average and maximum (95p) vapor pressure deficit at several locations.

2.5. Estimation of Snow Cover Presence

Snow cover fundamentally affects temperatures at soil surface. The insulating properties of snow cover and latent heat of melting cause temperature fluctuations under snow cover to be significantly dampened and in case of partially wet snow (which is the predominant state in Czech conditions) the temperature at the soil/snow interface stays close to 0°C. These principles can be used for detecting snow cover from microclimatic measurements. The `mc_calc_snow()` function serves for this, which detects snow presence at those measurements that meet conditions for maximum temperature (default value 1.25°C) and maximum temperature range (default value 1°C) in a moving time window (default value 3 days) on chosen temperature sensor (default choice ground sensor of Tomst TMS4 sensor). Reliability of snow detection with these default values is around 95% (by comparison with daily photos of location by camera trap). For deriving basic statistics (duration of snow cover, first/last day of snow cover and first/last day of continuous snow cover), the function `mc_calc_snow_agg()` serves. Continuous snow cover can be user-defined (default value 3 days).

```
## Snow detection based on ground temperatures (TMS_T2)
## Output saved as virtual sensor "snih"
micro.snow <- mc_calc_snow(micro.data,
                           sensor = "TMS_T2",
                           output_sensor = "snih",
                           range = 1,
                           tmax = 1.25,
                           days = 3)
## Visualization
mc_plot_line(micro.snow[c("CZ2_BUKOVEC",
                           "CZ2_KRKLAB",
                           "CS_26")], sensors = c("TMS_T2","snih"))
```

```
## Calculation of basic snow cover statistics for winter season 2021/2022
## Selection of data for chosen period
micro.snow.2021 <- mc_prep_crop(micro.snow,
                                start = as.POSIXct("2021-10-01", tz="UTC"),
                                end = as.POSIXct("2022-06-01", tz="UTC"))

## Calculation of snow cover duration,
## continuous snow cover defined as minimum of 7 consecutive days
tabulka.snih.2021 <- mc_calc_snow_agg(micro.snow.2021,
                                      snow_sensor = "snih",
                                      period = 7)
```

Table 7: Duration of snow cover automatically detected from temperature progression

locality_id	snow_days	first_day	last_day	first_day_period	last_day_period
CS_26	0	NA	NA	NA	NA
CZ2_BARINY	0	NA	NA	NA	NA
CZ2_BUKOVEC	153	2021-11-24	2022-04-25	2021-11-24	2022-04-25
CZ2_CEPICKA	40	2021-12-04	2022-02-06	2021-12-04	2022-02-06
CZ2_HODDUB	0	NA	NA	NA	NA
CZ2_HRUSUT	5	2022-01-25	2022-01-29	NA	NA
CZ2_KLINOV	119	2021-12-04	2022-05-01	2021-12-04	2022-05-01
CZ2_KRKLAB	165	2021-11-30	2022-05-13	2021-11-30	2022-05-13
CZ2_LUZNICE	0	NA	NA	NA	NA
CZ2_MASTALE	0	NA	NA	NA	NA

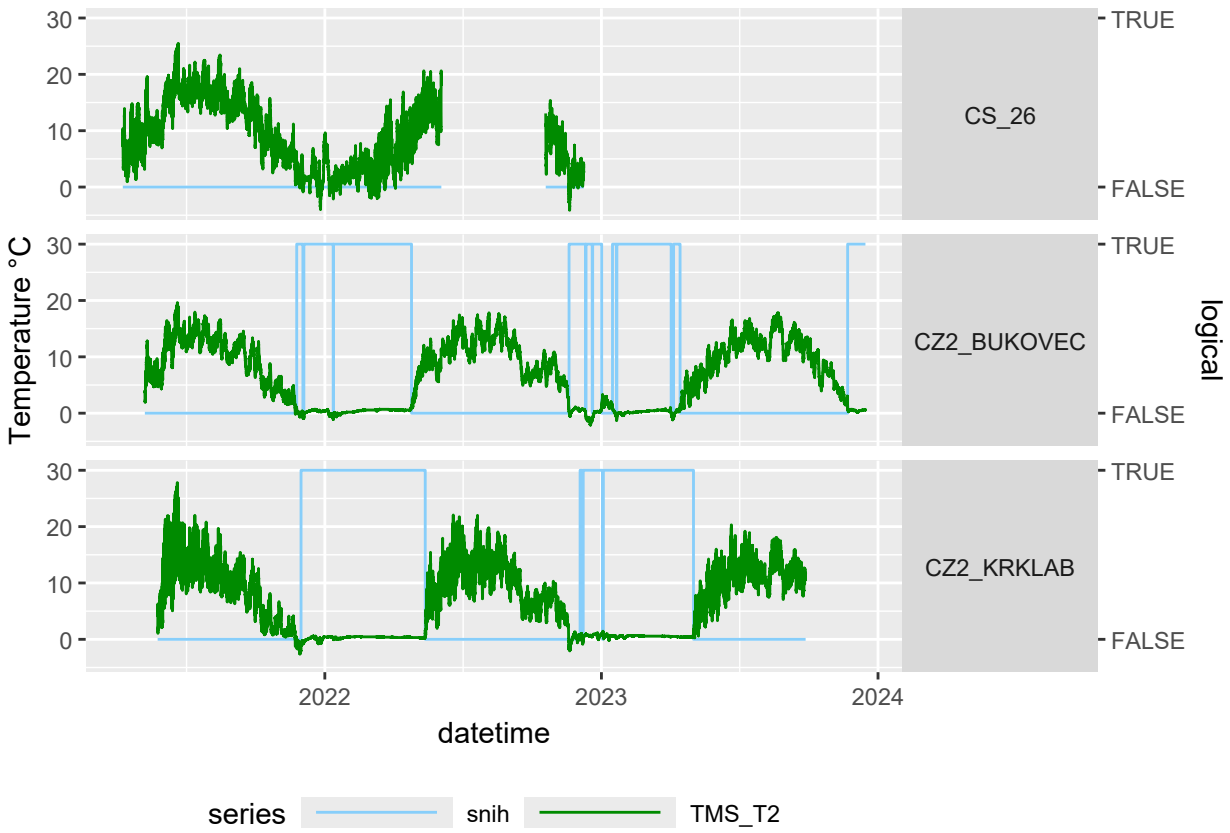

Figure 20: Visualization of detected snow on time series from several stations.

3. Evaluation and Comparison

We usually measure microclimate in the field with the aim of quantifying microclimatic differences between individual sites or to quantify the difference between the microclimate of a site of interest and a reference value for macroclimate, for example from station measurements of the Czech Hydrometeorological Institute (CHMI). The following examples demonstrate how to work with the data and how to verify whether microclimates differ.

3.1 Effect of Tree Species on Understory Microclimate

Introduction

A commonly used approach is to look for significant differences in microclimate between categories. These can be, for example, different tree species, habitat types, various disturbances, etc. Typically, at least 6-10 replicates per category are used if you expect a medium effect and have relatively low variability. If variability is higher or you expect a smaller effect, more independent replicates (10-20) may be needed. In addition to differences between categories, the difference from the control is also useful. In the case of disturbances, mature forest is a good control category. If we are interested in the influence of individual tree species, the control category can be in an open area.

Ideally, individual microclimatic stations should be independent of each other, for example, they should not be too close together. When monitoring the influence of individual trees, the minimum distance corresponds to the longest shadow of the given tree. Apart from the monitored categories, individual locations should not differ in other environmental characteristics - for example, altitude, exposure, shading, etc.

The following example draws from data obtained in 2024 in Průhonice Park. The aim was to compare the influence of conifers on understory microclimate. To eliminate the influence of topography, which creates differences in daily averages of up to 12°C within Průhonice Park (Brůna et al., 2023), we established the experiment within one topoclimatic unit "moderate northern slopes" (Wild et al., 2014).

We found the following species in sufficient numbers and quality to evaluate their influence on microclimate: *Abies alba, Abies grandis, Picea abies* and *Pseudotsuga menziesii*. We selected individual trees 5-9 meters tall, with 8 replicates for each tree species. A TOMST TMS-4 microclimatic station was placed on the north side of each tree trunk. To verify the influence of conifer stands on microclimate in general, 8 microclimatic stations were placed within a nearby young plantation in an open unshaded area.

Methods

We begin by loading information about files from the files_table.csv table, which needs to be prepared according to the information in chapter 1.4

```
## Loading 'files_table'
ft <- read.table("./data/dreviny/files_table.csv", sep=";", header = TRUE)
# For loading from subfolder, we need to modify the path column (relative path to current folder)
ft$path<-paste0("./data/dreviny/data/",ft$path)
```

```
# Loading logger data with metadata
tms <- mc_read_data(files_table = ft, silent = TRUE)
```

Measurements were conducted from December 15, 2023 to July 22, 2024. For this example, we selected spring March 21 - June 20, 2024. For this period, we calculated average, minimum and maximum temperature and volumetric soil moisture. For calculations using soil moisture, it is important to correctly set the soil type at each site, or alternatively, raw sensor data can be used if the soil is the same everywhere.

```r
## Cropping time series to selected period - spring
start <- as.POSIXct("2024-03-21", tz = "UTC")
end <- as.POSIXct("2024-06-21", tz = "UTC")
tms.jaro <- mc_prep_crop(tms, start, end, end_included=FALSE)
filename_prefix<-"jaro_"

## Calculation of virtual sensor VWC from raw TMS moisture data
tms.jaro <- mc_calc_vwc(tms.jaro, soiltype = "loamy sand A")

## Calculation of growing and frost degrees (GDD and FDD)
tms.jaro <- mc_calc_gdd(tms.jaro, sensor = "TMS_T3")
tms.jaro <- mc_calc_fdd(tms.jaro, sensor = "TMS_T3")

## Aggregation of all time series, calculation of means, variances
tms.jaro.vse <- mc_agg(tms.jaro, fun = c("mean", "range", "coverage",
                                         "percentile", "min", "max", "sum"),
                       percentiles = 95, period = "all", min_coverage = 0.95)

## Conversion to long format (for better data manipulation)
tms.jaro.vse.long <- mc_reshape_long(tms.jaro.vse)

## Conversion to wide format for easier analysis
df_wide.jaro <- tms.jaro.vse.long %>%
  select(-height) %>%
  pivot_wider(names_from = sensor_name, values_from = value)

## Loading site data and removing site "Plaste_40"
lokality_df <- read.csv("./data/dreviny/lokality.csv", sep=";")
lokality_df <- lokality_df[lokality_df$locality_id != "Plaste_40",]
# Joining with data
df_wide.jaro <- lokality_df %>%
  left_join(df_wide.jaro, by = "locality_id")
```

In summer 2024, hemispheric photographs of all sites were taken at a height of 80 cm above ground and canopy cover was derived using the HemisphereR library (Chianucci and Macek 2023). Cover was evaluated for the entire hemisphere and for individual 10° sectors. This allows us to verify microclimate differences caused by canopy cover. From field notes, we know that site Plaste_40 was destroyed and data from it is not available, so we will remove it from the site table as well.

In the site table, we also have a column "drevina" (tree species) which indicates under which tree each TMS-4 station was placed. Similarly, other relevant variables for analysis can be listed here, such as altitude, coordinates, slope, species presence, etc.

Table 8: Preview of site table including metadata.

locality_id	lat_wgs84	lon_wgs84	drevina	GapFraction	Openness
Plaste_01	49.99036	14.56065	Picea abies	4.97	4.97
Plaste_02	49.99040	14.56075	Abies alba	5.08	5.05

For visualization, we will create box plots that show the median, quartile range, and outliers of measured variables for individual species and control.

Statistical significance at the 95% confidence level will be determined by analysis of variance with post-hoc test (Tukey HSD) for comparing individual species and control (more in Crawley 2013). The significance of

differences is indicated directly in the graph using letters (CLD, Compact letter display). Boxplots with the same letter are not significantly different, while those that don't share any letter are significantly different. Differences that are not significant may be due to variability in the data. To increase the chance of detecting significant differences, the number of independent measurements needs to be increased.

Results

First, we'll generate a graph for one variable - average underground temperature (TMS_T1_mean).

```
# Variable for analysis
promenna <- "TMS_T1_mean"

# ANOVA
model <- aov(as.formula(paste(promenna, "~ drevina")), data = df_wide.jaro)
tukey <- TukeyHSD(model)

# Compact letter display (CLD)
cld <- multcompLetters4(model, tukey)$drevina
cld_df <- data.frame(drevina = names(cld$Letters), Letters = cld$Letters)

# Modification of species names for two-line display
df_wide.jaro$drevina <- gsub(" ", "\n", df_wide.jaro$drevina)

# Joining with maximum value for correct letter placement in graph
max_values <- df_wide.jaro %>%
  group_by(drevina) %>%
  summarize(max_value = max(.data[[promenna]], na.rm = TRUE))
cld_df <- merge(cld_df, max_values, by = "drevina")

# Creating boxplot
ggplot(df_wide.jaro, aes(x = drevina, y = .data[[promenna]], fill = drevina)) +
  geom_boxplot() +
  geom_text(
    data = cld_df,
    aes(label = Letters, y = max_value + 0.1 * max_value),
    position = position_dodge(width = 0.75),
    vjust = -0.5
  ) +
  labs(
    x = "Species",
    y = promenna
  ) +
  theme_minimal() +
  theme(legend.position = "none", axis.text.x = element_text(face = "italic"))
```

To explore differences in individual climatic variables between species, it's useful to generate graphs for individual variables. The following code generates files into the data/dreviny/vysledky folder.

```
# List of variables for analysis - a graph will be created for each
variables <-
  c("TMS_T1_mean","TMS_T3_mean","TMS_T3_min",
    "TMS_T3_max", "TMS_T3_range","TMS_T3_percentile95",
    "VWC_moisture_mean","GDD5_sum","FDD0_sum","Openness")
```

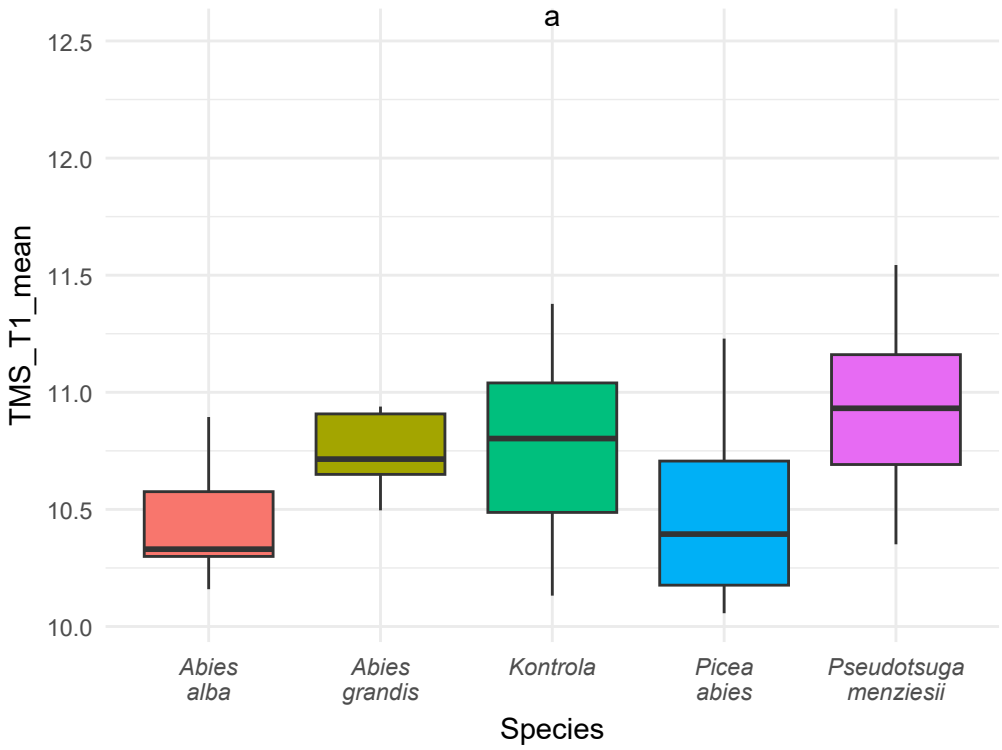

Figure 21: Boxplot for individual species with average underground temperature in spring.

```r
# Function for performing ANOVA, Tukey HSD test and creating boxplots
analyzuj_data <- function(df, promenna, prefix) {

# ANOVA
  model <- aov(as.formula(paste(promenna, "~ drevina")), data = df)
  tukey <- TukeyHSD(model)

# Compact letter display (CLD)
  cld <- multcompLetters4(model, tukey)$drevina
  cld_df <-data.frame(drevina = names(cld$Letters),
            Letters = cld$Letters)

# Joining with maximum value for correct letter placement in graph
  max_values <- df %>%
    group_by(drevina) %>%
    summarize(max_value = max(.data[[promenna]], na.rm = TRUE))
  cld_df <- merge(cld_df, max_values, by = "drevina")

# Creating boxplot
  p <-
    ggplot(df, aes(x = drevina, y = .data[[promenna]], fill = drevina)) +
    geom_boxplot() +
    geom_text(
      data = cld_df,
      aes(label = Letters, y = max_value + 0.1 * max_value),
```

```
      position = position_dodge(width = 0.75),
      vjust = -0.5
    ) +
    labs(
      title = paste("Boxplot", prefix, promenna),
      x = "Species",
      y = promenna
    ) +
    theme_minimal() +
    theme(legend.position = "none",
          axis.text.x = element_text(face = "italic"))

# Export boxplot in .png format
  ggsave(paste0( "./data/dreviny/vysledky/",
      prefix, promenna,"_boxplot.png"),
    plot = p, width = 8, height = 6 , bg = "white")
}

for (promenna in variables) {
  analyzuj_data(df_wide.jaro, promenna, filename_prefix)
}
```

In this example, the graphs were generated into files that you can find in the files section in the data/dreviny/vysledky folder. Below they are inserted directly as images.

The canopy cover of individual species did not differ significantly in any of the monitored parameters, while the control was always different. Differences in microclimate can thus be attributed mainly to differences between individual species.

Most climatic parameters at 15 cm above ground were significantly different at control sites, the influence of conifer stands on microclimate is undeniable.

Individual trees can buffer temperature extremes 15 cm above ground. Within spring values, this means reducing maximum temperatures by up to 5°C and increasing minimum temperatures by 2°C.

Temperature parameters in soil did not differ significantly between species and in most cases not even between forest and control.

The differences in temperature parameters detected at 15 cm above ground between species are not large and were significantly different in only a few parameters. The biggest differences were in maximum temperatures.

Maximum temperatures in spring under *Abies alba* trees differ significantly from those under *Pseudotsuga menziesii* trees and from control. Temperatures under all species differ significantly from control in spring.

In other parameters there are no significant differences, but a higher buffering effect is still visible, particularly for *Abies alba* and *Picea abies*. Under *Abies grandis* and *Pseudotsuga menziesii* trees, temperatures are often higher.

Volumetric soil moisture is significantly different only between control and species, with no significant difference in this parameter between individual species. At control sites during spring, soil moisture is 15 percentage points higher. This is probably mainly due to rainfall interception by tree branches and water use by tree roots.

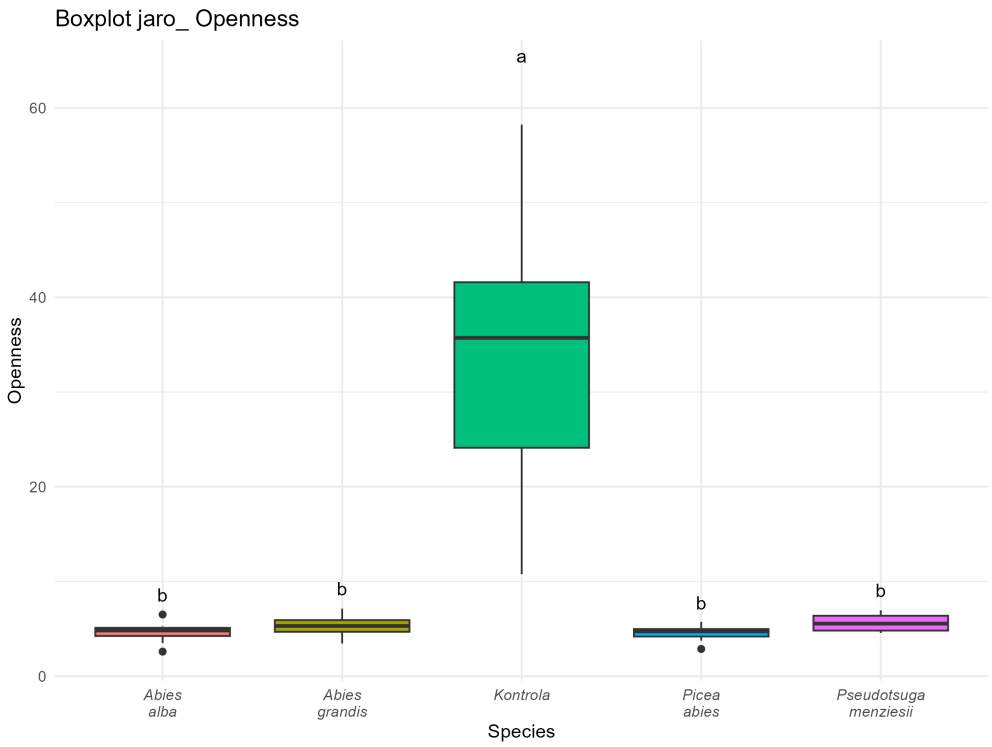

Figure 22: Forest stand openness calculated from hemispheric photographs serves to verify that locations differ only by species and that there is no difference in shading.

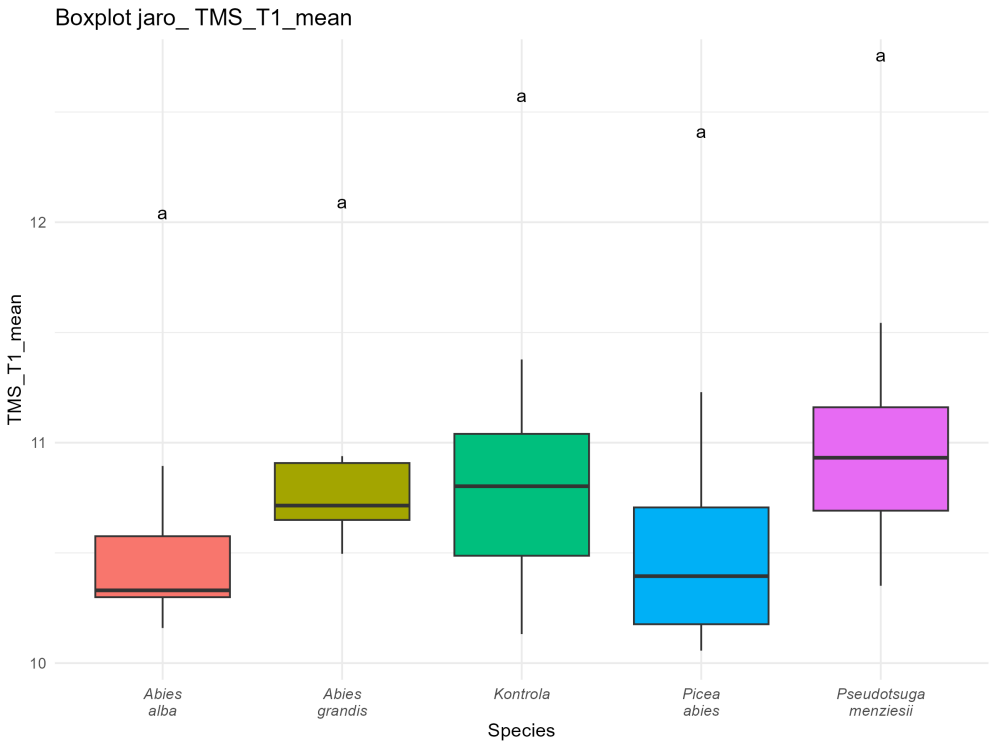

Figure 23: Boxplots of average soil temperatures by individual tree species.

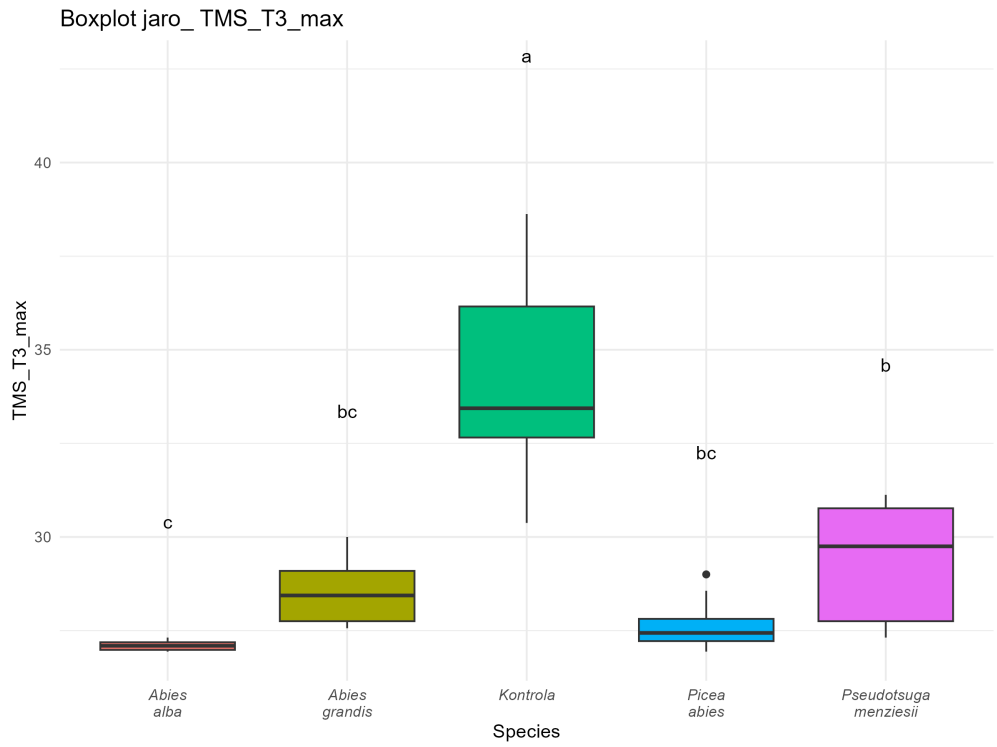

Figure 24: Boxplots of maximum spring temperatures at 15 cm above ground for locations divided by individual species.

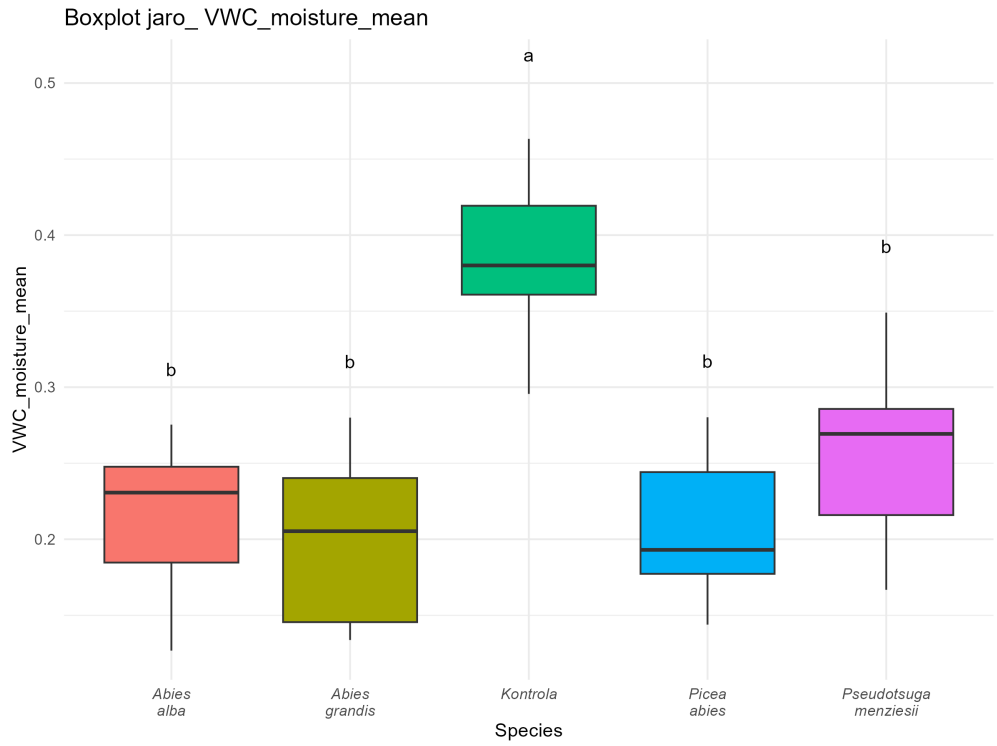

Figure 25: Boxplots of average volumetric soil moisture for locations divided by tree species during spring.

3.2. Comparison with CHMI Meteorological Stations

Introduction

We are often interested in whether the microclimate of our location differs from the nearest meteorological station. In June 2024, CHMI published daily data from their stations in machine-readable JSON format. The data also contains historical data in the form of daily averages. For temperatures, this is the SYNOP average, which is calculated from values at 6:00, 13:00, and 20:00 UTC. For simplicity, we will compare these values with daily averages of all measurements from TMS stations.

For examples in sections 3.2 and 3.3, we are loading data from a prepared file, since otherwise it would be necessary to download several GB of data and filter them. At the end of section 3.3, we include a script that we used to obtain and modify this data. When looking for a relevant CHMI station, it is necessary to consider not only its distance but also similar altitude and whether it measures the required variable. If the altitude was significantly different, it can be assumed that a large part of the temperature difference is caused mainly by this difference.

Methods

For two locations in the Šumava Mountains, we will find the nearest meteorological station. We will use a prepared file that contains only stations that measured temperature in 2022 and also have calculated distance from our station. By modifying the script at the end of section 3.3, you can download data for other stations according to the required variable. The nearest stations are at Horská Kvilda with WSI: 0-203-0-11494 and at Churáňov, with WSI code: 0-20000-0-11457. Churáňov has a more similar altitude and we will select only this station for further processing.

```
# Find nearest stations within given radius - WGS84 coordinates and radius in kilometers
stanice_s_informacemi<-readRDS("./data/stanice_s_informacemi.rds")
max_vzdalenost<-5 # 5 km
stanice_v_blizkosti<-stanice_s_informacemi[stanice_s_informacemi$vzdalenost_km
                                           <max_vzdalenost,]
print(stanice_v_blizkosti$WSI)
```

```
## [1] "0-20000-0-11457" "0-203-0-11494"
```

```
# Select only Churáňov, as it has more similar altitude
stanice_v_blizkosti<-stanice_s_informacemi[stanice_s_informacemi$WSI=="0-20000-0-11457",]

# Loading prepared data
climate_data<-readRDS("./data/chmu_data_2022.rds")

# Filtering climate_data based on corresponding WSI values
climate_data <- climate_data[climate_data$STATION %in% stanice_v_blizkosti$WSI, ]
```

Now we will load our 2 locations - this time in TMS_join format - in this form we often provide processed data that may come from a combination of sensors that were at the location over several years. In this example, we use data from a Thermologger placed at a height of 2 m above ground on the north side of a tree.

```
# Loading TOMST Thermologger files using files_table
ft <- read.table("./data/stanice/files_table.csv", sep=";", header = TRUE)
ft$path<-paste0("./data/stanice/",ft$path)
```

```r
# Loading logger data with metadata
tms <- mc_read_data(files_table = ft, silent = TRUE)

# Cropping time series to selected period
start <- as.POSIXct("2022-01-01", tz = "UTC")
end <- as.POSIXct("2023-01-01", tz = "UTC")
tms <- mc_prep_crop(tms, start, end, end_included=FALSE)

# Aggregation of all time series, calculation of means, variances
tms.agg <- mc_agg(tms, fun = c("mean"), period = "day", min_coverage = 0.95)

# Conversion to long format (for better data manipulation)
tms.agg.long <- mc_reshape_long(tms.agg)

## Conversion to wide format for easier analysis
df_wide.tms <- tms.agg.long %>%
  select(-height) %>%
  pivot_wider(names_from = sensor_name, values_from = value)

# Saving for later use (here saving a table, so saveRDS is sufficient and
# no need to call myClim::mc_save)
saveRDS(df_wide.tms, file = "./data/dve_tms_sumava.rds")
```

Results

Now we will create a simple graph with data from the Churáňov station and from our microclimatic stations.

```r
# Combining datasets into one with added 'locality' column
combined_data <- rbind(
  data.frame(Date = climate_data$DT,
             Temp = as.numeric(climate_data$VAL), Locality = "CHMI"),
  data.frame(Date = as.Date(df_wide.tms[df_wide.tms$locality_id==
                                        "NPS_4210_D_T1",]$datetime),
             Temp = df_wide.tms[df_wide.tms$locality_id=="NPS_4210_D_T1",]$Thermo_T_mean,
             Locality = "NPS_4210_D_T1"),
  data.frame(Date = as.Date(df_wide.tms[df_wide.tms$locality_id=="4220_TMS",]$datetime),
             Temp = df_wide.tms[df_wide.tms$locality_id=="4220_TMS",]$Thermo_T_mean,
             Locality = "4220_TMS")
)

# Using a color palette easily distinguishable for colorblind viewers (Okabe-Ito palette)
ggplot(combined_data, aes(x = Date, y = Temp, color = Locality)) +
  geom_line() +
  scale_color_manual(values = c("#E69F00", "#000000", "#009E73")) +
  labs(title = "Average daily temperature from three sources",
       x = "Date", y = "Temperature (°C)",
       color = "Location") +
  theme_minimal()
```

The large difference in March is particularly interesting, which we will examine further.

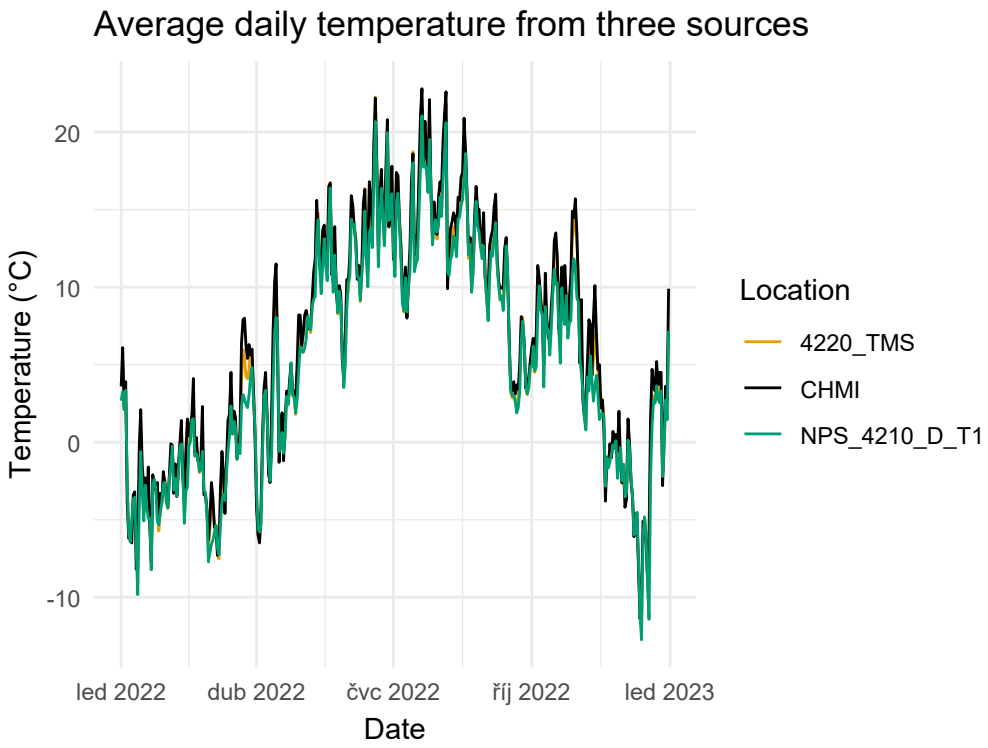

Figure 26: At first glance, it is evident that average temperatures have similar patterns and that at TMS stations they are usually lower.

```
# Define data range for selection
start_date <- as.Date("2022-03-01")
end_date <- as.Date("2022-03-31")

filtered_data <- combined_data[combined_data$Date >= start_date
                               & combined_data$Date <= end_date,]

# Creating graph for selection
ggplot(filtered_data, aes(x = Date, y = Temp, color = Locality)) +
  geom_line() +
  scale_color_manual(values = c("#E69F00", "#000000", "#009E73")) +
  labs(title = "Average daily temperature from three sources",
       x = "Date", y = "Temperature (°C)",
       color = "Location"
      ) +
  theme_minimal()
```

For easier evaluation, we can look at a histogram of temperature differences between each microclimatic station and the Churáňov station.

```
# Joining datasets GSOD, NPS_4210_D_T1 and 4220_TMS by date
merged_data <- combined_data %>%
  spread(Locality, Temp)

# Calculation of daily differences
```

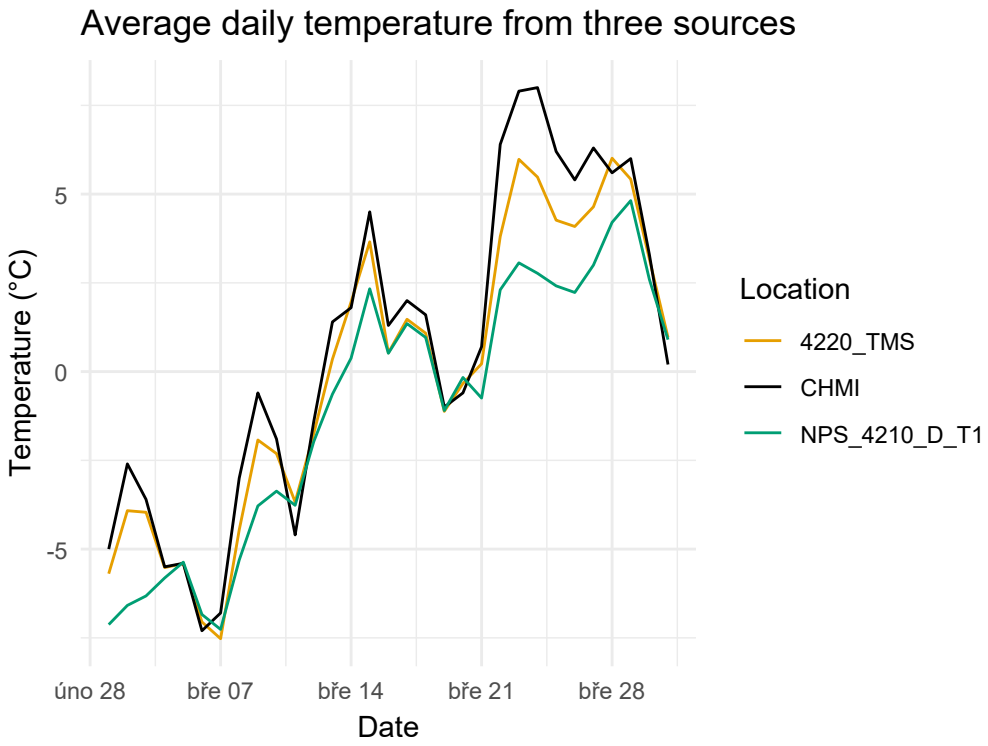

Figure 27: In some days, average temperatures differed by up to 10°C. During warm days, average temperatures at TMS stations were lower, while during colder days, average temperatures at TMS stations were higher.

```r
merged_data <- merged_data %>%
  mutate(Difference_NPS_CHMU = NPS_4210_D_T1 - CHMI,
         Difference_TMS_CHMU = `4220_TMS` - CHMI)

# Getting x and y axis range for both graphs
x_limits <- range(c(merged_data$Difference_NPS_CHMU,
                    merged_data$Difference_TMS_CHMU),
            na.rm = TRUE)

y_limits <- c(0, max(table( cut(merged_data$Difference_NPS_CHMU,
            breaks = seq(min(x_limits), max(x_limits), by = 0.5))),
            table(cut(merged_data$Difference_TMS_CHMU,
            breaks = seq(min(x_limits), max(x_limits), by = 0.5)))))

# Histogram for difference between NPS_4210_D_T1 and GSOD
p1 <- ggplot(merged_data, aes(x = Difference_NPS_CHMU)) +
  geom_histogram(binwidth = 0.5, fill = "#E69F00",
                 color = "black") +
  geom_vline(xintercept = 0, linetype = "dashed",
             color = "red") +  # Adding line at x=0
  scale_x_continuous(limits = x_limits) +  # Same x axis for both graphs
  scale_y_continuous(limits = y_limits) +  # Same y axis for both graphs
  labs(title = "NPS_4210_D_T1 - CHMI",
       x = "Temperature difference (°C)", y = "Number of days") +
  theme_minimal()

# Histogram for difference between 4220_TMS and CHMI
p2 <- ggplot(merged_data, aes(x = Difference_TMS_CHMU)) +
  geom_histogram(binwidth = 0.5, fill = "#56B4E9",
                 color = "black") +
  geom_vline(xintercept = 0, linetype = "dashed",
             color = "red") +  # Adding line at x=0
  scale_x_continuous(limits = x_limits) +  # Same x axis for both graphs
  scale_y_continuous(limits = y_limits) +  # Same y axis for both graphs
  labs(title = "4220_TMS - CHMI",
       x = "Temperature difference (°C)", y = "Number of days") +
  theme_minimal()

# Displaying graphs side by side
grid.arrange(p1, p2, ncol = 2)
```

A scatterplot can be a good visualization tool, where we see station data on the X axis and microclimatic station data on the Y axis.

```r
# Getting minimum and maximum values for X and Y axes (including 0 and same limits for both axes)
min_limit <- min(c(merged_data$CHMI,
                   merged_data$NPS_4210_D_T1,
                   merged_data$`4220_TMS`),
                 na.rm = TRUE)
max_limit <- max(c(merged_data$CHMI,
                   merged_data$NPS_4210_D_T1,
                   merged_data$`4220_TMS`),
                 na.rm = TRUE)
```

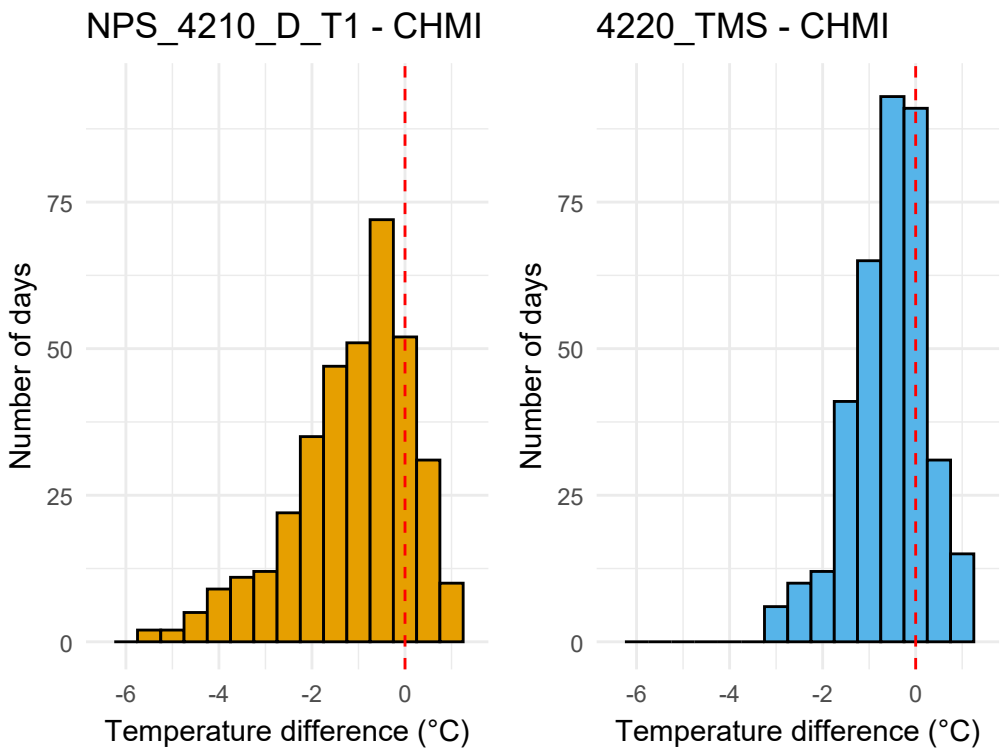

Figure 28: Histogram of differences in average daily temperatures between individual TMS stations and average daily temperatures from the CHMI station at Churáňov.

```
axis_limits <- c(min(0, min_limit), max_limit)

# Creating scatterplot
ggplot(merged_data, aes(x = CHMI)) +
  # Points for NPS_4210_D_T1 with transparency
  geom_point(aes(y = NPS_4210_D_T1, color = "NPS_4210_D_T1"), alpha = 0.6) +
  # Points for 4220_TMS with transparency
  geom_point(aes(y = `4220_TMS`, color = "4220_TMS"), alpha = 0.6) +
  # 1:1 line
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
  # Black line at Y = 0
  geom_hline(yintercept = 0, color = "black", size = 0.5) +
  # Black line at X = 0
  geom_vline(xintercept = 0, color = "black", size = 0.5) +
  # Same limits for X and Y axes
  scale_x_continuous(limits = axis_limits) +
  scale_y_continuous(limits = axis_limits) +
  labs(title = "Comparison of CHMI stations with NPS_4210_D_T1 and 4220_TMS",
       x = "CHMI Temperature (°C)", y = "Temperature (°C)",
       color = "Location") +  # Legend label
  theme_minimal() +
  scale_color_manual(values = c("NPS_4210_D_T1" = "#E69F00",
                                "4220_TMS" = "#56B4E9")) +
  theme(axis.line = element_blank(),)  # Removing axis lines
```

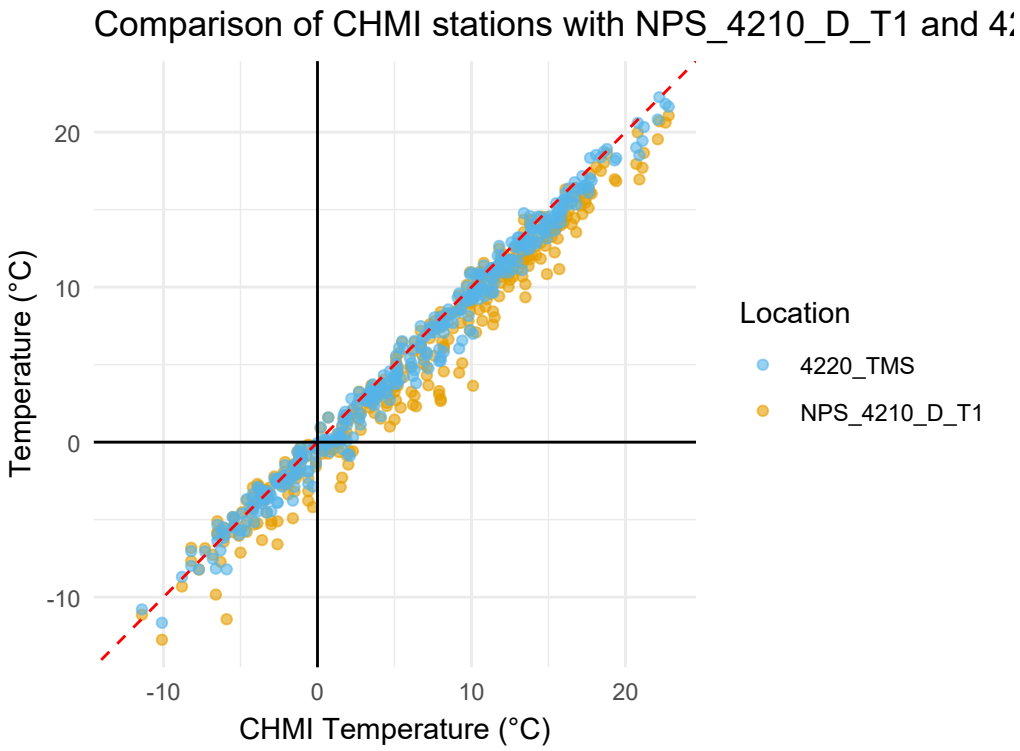

Figure 29: When using a scatterplot, it's good to observe the position relative to the 1:1 line. This allows us to see that during both cold and warm days, TMS station averages were generally lower, but higher values are not uncommon, even in the cold part. At station NPS_4210_D_T1, averages are mostly lower than Churáňov data.

For easier presentation of results, a simple calculation of average daily temperature difference is useful. In microclimate studies, the "buffering effect" of the location is often used.

```
# Calculation of average difference (buffering effect) and standard deviation
buffering_effect <- merged_data %>%
  summarise(
    Mean_Buffering_NPS_4210_D_T1 = mean(Difference_NPS_CHMU, na.rm = TRUE),
    SD_Buffering_NPS_4210_D_T1 = sd(Difference_NPS_CHMU, na.rm = TRUE),
    Mean_Buffering_4220_TMS = mean(Difference_TMS_CHMU, na.rm = TRUE),
    SD_Buffering_4220_TMS = sd(Difference_TMS_CHMU, na.rm = TRUE)
  )

# Creating text output for both locations
cat(sprintf("Vegetation buffering effect at location NPS_4210_D_T1 is %.2f ± %.2f °C\n",
            buffering_effect$Mean_Buffering_NPS_4210_D_T1,
            buffering_effect$SD_Buffering_NPS_4210_D_T1))
```

```
## Vegetation buffering effect at location NPS_4210_D_T1 is -1.13 ± 1.33 °C
```

```
cat(sprintf("Vegetation buffering effect at location 4220_TMS is %.2f ± %.2f °C\n",
            buffering_effect$Mean_Buffering_4220_TMS,
            buffering_effect$SD_Buffering_4220_TMS))
```

```
## Vegetation buffering effect at location 4220_TMS is -0.57 ± 0.81 °C
```

It is evident that the effect is not very large and has a high standard deviation. Testing the significance of such a difference is more complex. The commonly used paired T-test assumes that differences between pairs are independent and have a normal distribution. If individual days are not independent (for example, temperatures between consecutive days may be strongly correlated due to seasonal and daily trends), the assumption of independence may be violated, which will affect the test results. This can lead to:

- **underestimation of variability**: If values are dependent (e.g., temperatures on consecutive days), this can lead to underestimation of variability, which may cause the test to falsely detect statistically significant differences.
- **higher probability of Type I error**: There may be a higher probability of Type I error, meaning we reject the null hypothesis (that there is no difference) even when it is true.

How to proceed?

The correct procedure is then to use ARIMA models for example **(AutoRegressive Integrated Moving Average)** (Box et Jenkins, 1970). These models are specially designed to capture patterns in time series where values between individual days are dependent on previous values.

How ARIMA works:

- **AR (AutoRegressive)**: The AR part of the model assumes that the current value is a linear combination of previous values (lags).
- **I (Integrated)**: Helps remove trend or seasonal component through differencing (subtracting neighboring values).
- **MA (Moving Average)**: MA model takes into account dependence on previous model errors.

However, this is beyond the scope of this methodology.

3.3. Comparison with Stations Across the Czech Republic

Introduction

Here we will again use the GSODR library, but we'll look at how our stations compare to temperatures across the Czech Republic along an elevation gradient.

Methods

We'll select one day and download data from stations in the Czech Republic for that day (data is downloaded by whole years, then filtered)

```
# Enter date of interest
date_of_interest <- as.Date("2022-09-01")

# Loading prepared data
stanice_s_informacemi<-readRDS("./data/stanice_s_informacemi.rds")
climate_data<-readRDS("./data/chmu_data_2022.rds")

# Joining station information with temperature data
climate_data <- climate_data %>%
  left_join(stanice_s_informacemi, by = c("STATION" = "WSI"),
            relationship = "many-to-many")

# Data adjustment
climate_data$VAL<-as.numeric(climate_data$VAL)
climate_data$ELEVATION<-as.numeric(climate_data$ELEVATION)

# Filtering data for specific date
filtered_data <- subset(climate_data, DT == date_of_interest)
```

And now we'll load data from the same microclimatic stations as in the previous example. Additionally, we'll add their elevation so we can compare them with stations in the Czech Republic in the graph and filter for the selected day.

```
# Loading data
df_wide.tms<-readRDS("./data/dve_tms_sumava.rds")
# Adding elevation
df_wide.tms$elevation <- 1120
df_wide.tms$mean_temp <- df_wide.tms$Thermo_T_mean
# Filtering for selected day
filtered_tms <- subset(df_wide.tms, as.Date(datetime) == date_of_interest)
```

Results

And now we'll create a graph for the selected day (September 1, 2022), where we'll also add a simple regression of temperature and elevation with confidence intervals.

```
# Check if data is available for given date
if (nrow(filtered_data) > 0) {
  # Create scatter plot temperature vs. elevation with trend line
  p <- ggplot(filtered_data, aes(x = ELEVATION, y = VAL)) +
```

```
    geom_point(color = "blue", alpha = 0.7) +
    # Adding regression line and confidence intervals
    geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed",
                fill = "lightblue", alpha = 0.3) +
    labs(title = paste("Temperature vs. elevation on", date_of_interest,
                       "source: CHMI"),
         x = "Elevation (m a.s.l.)", y = "Average daily temperature (°C)") +
    theme_minimal()

  # Adding data from logger stations as red points
  p <- p +
    geom_point(data = filtered_tms, aes(x = elevation, y = mean_temp),
               color = "red", size = 3)
  # Displaying graph
  print(p)
} else {
  cat("No data available for selected date:", date_of_interest, "\n")
}
```

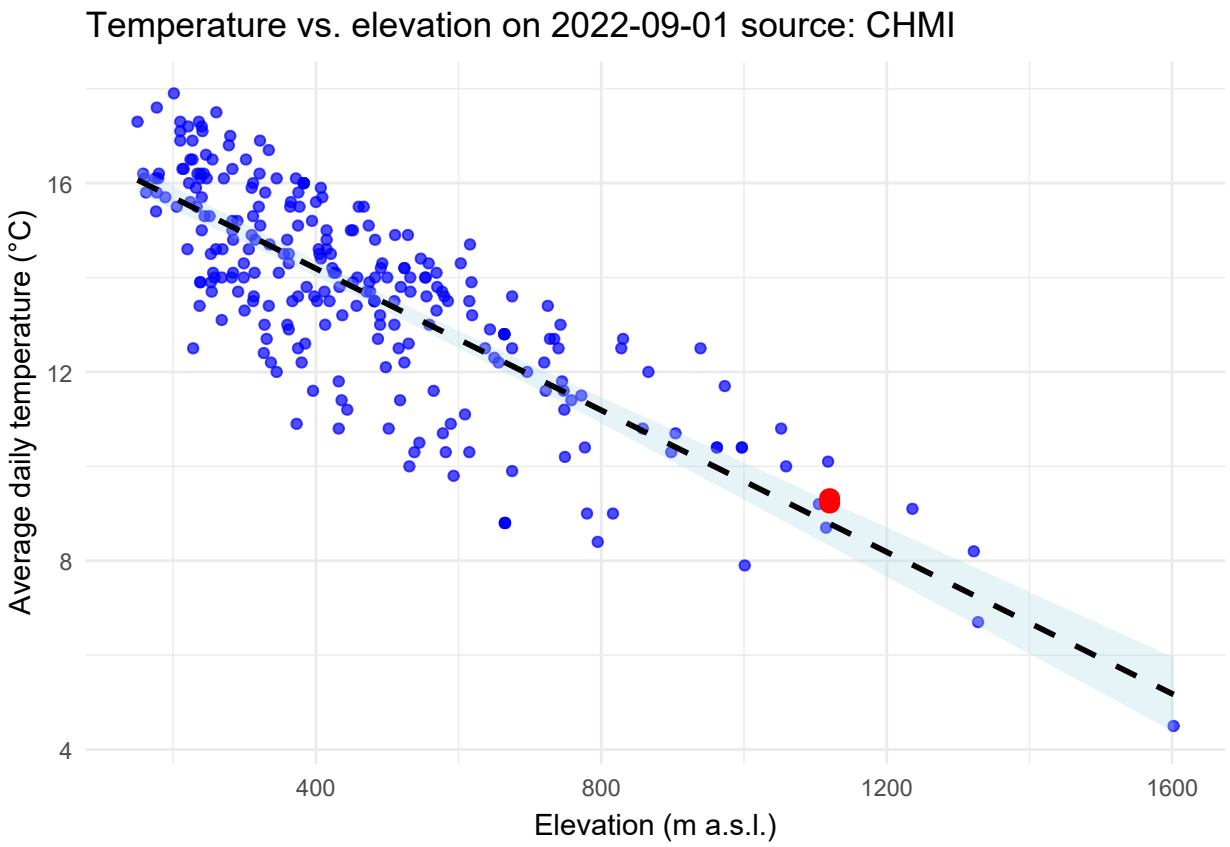

Figure 30: It can be seen that on September 1, 2022, the average temperature at the station did not deviate from values across the Czech Republic. However, the graph also shows that there are few stations in mountain locations.

And how was it on other days? We'll use the fact that we already have the data loaded, we just select (filter) a different day.

```r
# Enter date of interest
date_of_interest <- as.Date("2022-01-12")

# Filter data for specific date
filtered_data <- subset(climate_data, DT == date_of_interest)
filtered_tms <- subset(df_wide.tms, as.Date(datetime) == date_of_interest)

# Check if data is available for given date
if (nrow(filtered_data) > 0) {
  # Create scatter plot temperature vs. elevation with trend line
  p <- ggplot(filtered_data, aes(x = ELEVATION, y = VAL)) +
    geom_point(color = "blue", alpha = 0.7) +
    # Adding regression line and confidence intervals
    geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed",
                fill = "lightblue", alpha = 0.3) +
    labs(title = paste("Temperature vs. elevation on", date_of_interest,
                       "source: CHMI"),
         x = "Elevation (m a.s.l.)", y = "Average daily temperature (°C)") +
    theme_minimal()

  # Adding data from logger stations as red points
  p <- p +
    geom_point(data = filtered_tms, aes(x = elevation, y = mean_temp),
               color = "red", size = 3)
  # Displaying graph
  print(p)
} else {
  cat("No data available for selected date:", date_of_interest, "\n")
}
```

And what about during a temperature inversion? This occurred for example on October 31, 2022.

```r
# Date of interest
date_of_interest <- as.Date("2022-10-31")

# Filter data for specific date
filtered_data <- subset(climate_data, DT == date_of_interest)
filtered_tms <- subset(df_wide.tms, as.Date(datetime) == date_of_interest)

# Check if data is available for given date
if (nrow(filtered_data) > 0) {
  # Create scatter plot temperature vs. elevation with trend line
  p <- ggplot(filtered_data, aes(x = ELEVATION, y = VAL)) +
    geom_point(color = "blue", alpha = 0.7) +
    # Adding regression line and confidence intervals
    geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed",
                fill = "lightblue", alpha = 0.3) +
    labs(title = paste("Temperature vs. elevation on", date_of_interest,
                       "source: CHMI"),
         x = "Elevation (m a.s.l.)", y = "Average daily temperature (°C)") +
    theme_minimal()
  # Adding data from logger stations as red points
  p <- p +
    geom_point(data = filtered_tms, aes(x = elevation, y = mean_temp),
```

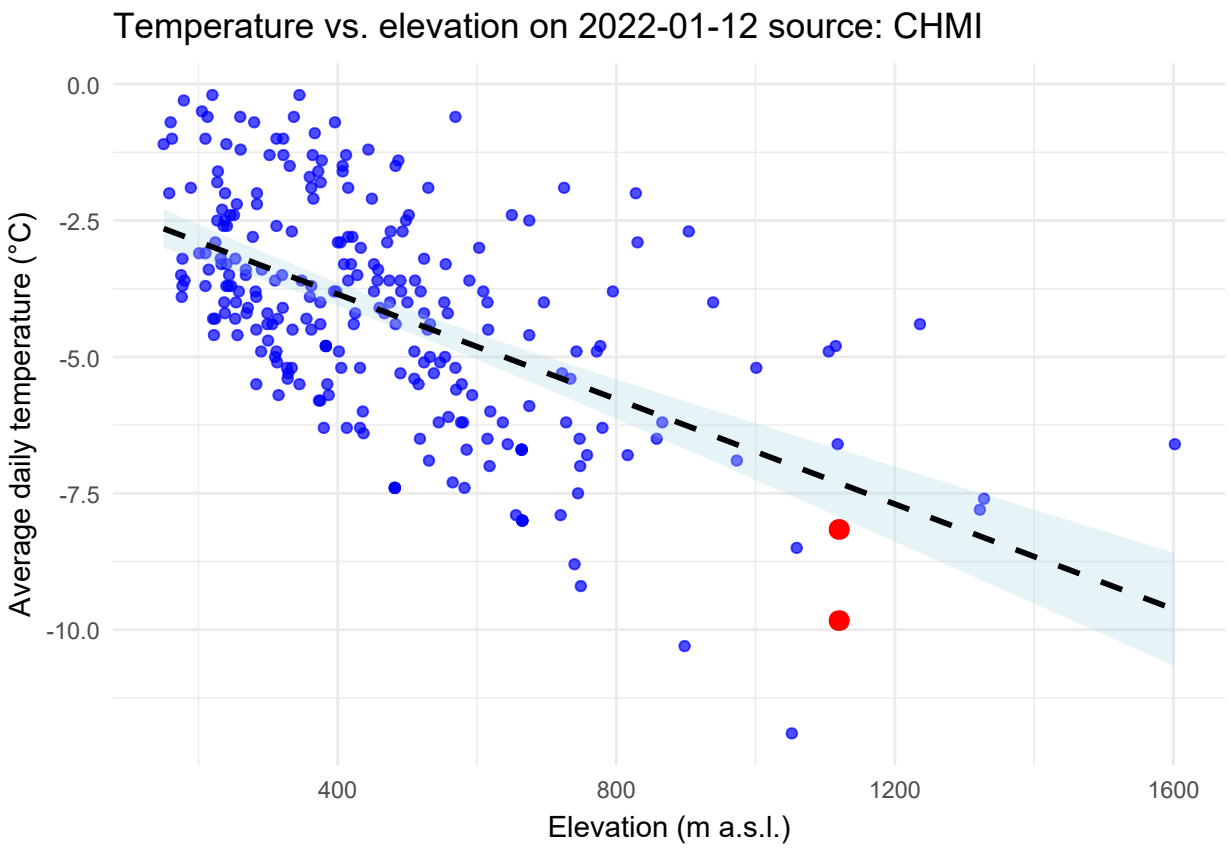

Figure 31: On January 12, 2022, the temperature at our stations was lower than we would expect based on stations in the Czech Republic at this elevation, which corresponds to the value at Churáňov. We can speculate whether this was due to the influence of snow in the forest or just simple shading during a sunny winter day.

```
                color = "red", size = 3)
  # Displaying graph
  print(p)
} else {
  cat("No data available for selected date:", date_of_interest, "\n")
}
```

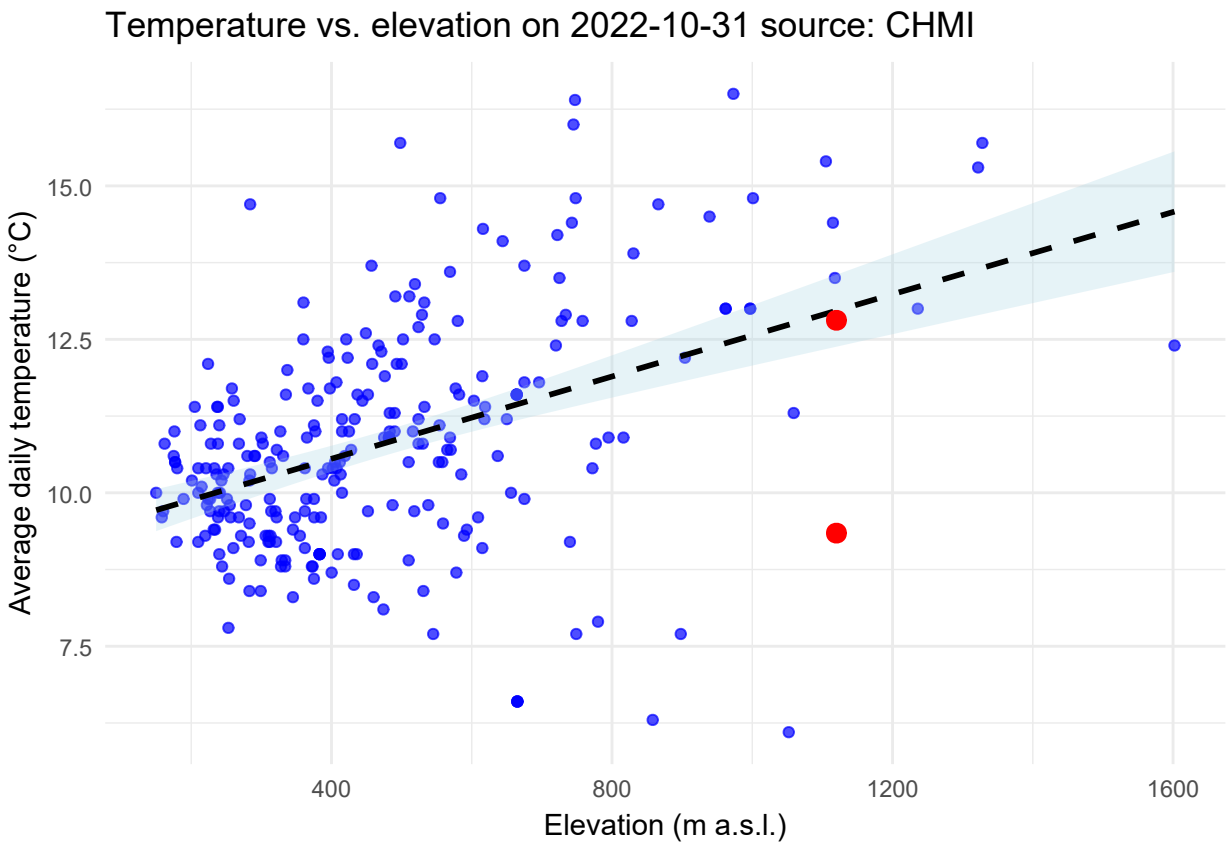

Figure 32: In this case too, the temperature at our stations was lower than would correspond to meteorological stations in open areas.

Script for Downloading and Filtering CHMI Data

The following script is what we used to prepare the materials for chapters 3.2 and 3.3. By modifying it, you can download data for other stations, variables, or time periods.

```
# List of required libraries
required_packages <- c("jsonlite", "lubridate", "sf", "dplyr", "stringr")

# Installation and loading of libraries
suppressWarnings(
  lapply(required_packages, function(pkg) {
    if (!require(pkg, character.only = TRUE)) install.packages(pkg,
                                repos = "https://mirrors.nic.cz/R/")
    library(pkg, character.only = TRUE)
```

```
  })
)
rm(required_packages)

# Function definitions
# Function for downloading metadata and filtering stations based on available variable and year
ziskat_stanice_s_dostupnou_promennou <- function(rok, element) {
  # Downloading metadata file meta2.json using curl
  metadata_url <-
    "https://opendata.chmi.cz/meteorology/climate/historical/metadata/meta2.json"
  metadata_file <- tempfile(fileext = ".json")
  system(paste("curl -o", metadata_file, metadata_url))
  # Loading metadata from file
  metadata <- fromJSON(metadata_file)
  # Processing metadata, extracting values and filtering by variable and year
  metadata_df <- data.frame(metadata$data$data$values)
  colnames(metadata_df) <- unlist(strsplit(metadata$data$data$header, ","))
  # Filtering stations that have available data for chosen year and variable (element)
  stanice_s_promennou <- metadata_df %>%
    filter(EG_EL_ABBREVIATION == element) %>%
    mutate(BEGIN_DATE = as.Date(BEGIN_DATE),
           END_DATE = as.Date(END_DATE),
           HEIGHT=as.numeric(HEIGHT)) %>%
    filter(year(BEGIN_DATE) <= rok & year(END_DATE) >= rok & HEIGHT < 2.5 & HEIGHT > 1.9)
  return(stanice_s_promennou)
}

# Function for loading meta1.json and joining station information with filtering by year
pripojit_meta1 <- function(stanice_s_promennou, rok) {
  # Downloading meta1.json using download.file
  metadata1_url <-
    "https://opendata.chmi.cz/meteorology/climate/historical/metadata/meta1.json"
  metadata1_file <- tempfile(fileext = ".json")
  download.file(metadata1_url, metadata1_file)
  metadata1 <- fromJSON(metadata1_file)
  # Extracting information from meta1.json and creating data frame
  metadata1_df <- data.frame(metadata1$data$data$values)
  colnames(metadata1_df) <- c("WSI", "GH_ID", "BEGIN_DATE", "END_DATE", "FULL_NAME",
                              "GEOGR1", "GEOGR2", "ELEVATION")
  # Converting BEGIN_DATE and END_DATE to Date format
  metadata1_df <- metadata1_df %>%
    mutate(BEGIN_DATE = as.Date(BEGIN_DATE),
           END_DATE = as.Date(END_DATE))
  # Filtering based on given year and ensuring GEOGR1 and GEOGR2 are not NA
  metadata1_df_filtered <- metadata1_df %>%
    filter(year(BEGIN_DATE) <= rok & year(END_DATE) >= rok) %>%
    filter(!is.na(GEOGR1) & !is.na(GEOGR2))
  # Joining information from meta1 to stanice_s_promennou table using left_join
  stanice_s_informacemi <- stanice_s_promennou %>%
    left_join(metadata1_df_filtered, by = "WSI")
  stanice_s_informacemi <- stanice_s_informacemi %>%
    filter(!is.na(GEOGR1) & !is.na(GEOGR2))
  return(stanice_s_informacemi)
```

```
}

# Function for calculating distance in kilometers in UTM33N
vypocet_vzdalenosti_utm33n <- function(lat1, lon1, lat2, lon2) {
  # Creating sf objects for points
  bod1 <- st_as_sf(data.frame(lon = lon1, lat = lat1),
                   coords = c("lon", "lat"), crs = 4326)
  bod2 <- st_as_sf(data.frame(lon = lon2, lat = lat2),
                   coords = c("lon", "lat"), crs = 4326)
  # Transforming coordinates to UTM33N (EPSG:32633)
  bod1_utm <- st_transform(bod1, 32633)
  bod2_utm <- st_transform(bod2, 32633)
  # Calculating distance in meters and converting to kilometers
  vzdalenost_m <- st_distance(bod1_utm, bod2_utm)
  vzdalenost_km <- as.numeric(vzdalenost_m) / 1000
  return(vzdalenost_km)
}

# Function for filtering stations based on distance from given point
filtrovat_stanice_dle_vzdalenosti <- function(stanice_df, lat, lon, max_vzdalenost_km) {
  # Calculating distance for each station
  stanice_df <- stanice_df %>%
    rowwise() %>%
    mutate(vzdalenost_km = vypocet_vzdalenosti_utm33n(lat, lon, as.numeric(GEOGR2),
                                                      as.numeric(GEOGR1))) %>%
    ungroup() %>%
    # Filtering by distance
    filter(vzdalenost_km <= max_vzdalenost_km)
  return(stanice_df)
}

# Function for loading and cleaning JSON file with jsonlite
ziskat_data_stanice_pro_rok <- function(stanice_id, element, vtype, rok) {
  # Folder for storing data
  dir.create("./data/stanice", recursive = TRUE, showWarnings = FALSE)
  filename <- paste0("./data/stanice/dly-", stanice_id, ".json")

  # If file exists, load it, otherwise download it
  if (!file.exists(filename)) {
    # Modified URL format for downloading without specifying year
    data_url <-
      paste0("https://opendata.chmi.cz/meteorology/climate/historical/data/daily/dly-", stanice_id, ".j
    # Download file using curl
    download_command <- paste("curl -w '%{http_code}' -o", shQuote(filename), data_url)
    status_code <- system(download_command, intern = TRUE)

    # Ensure we check only first value of status_code
    if (length(status_code) == 0 || status_code[1] != "200") {
      message("File is not available or cannot be downloaded (HTTP status: ", status_code, "): ", data_
      return(NULL)
    }
  }
```

```
  # Attempt to load JSON data using jsonlite with error handling
  data <- tryCatch({
    fromJSON(filename, simplifyVector = TRUE)  # Loading JSON file using jsonlite
  }, error = function(e) {
    # If error occurs, print error and skip file
    message("Error loading JSON file using jsonlite: ", filename, " - ", e$message)
    return(NULL)  # Skip this file and return NULL
  })

  if (is.null(data)) {
    return(NULL)  # If JSON file wasn't loaded, skip file
    }

  # Check if JSON contains data
  if (!"data" %in% names(data)) {
    message("Missing data in downloaded file: ", filename)
    return(NULL)
  }

  # Attempt to convert data to data frame
  data_filtered <- tryCatch({
    as.data.frame(data$data$data$values)
  }, error = function(e) {
    message("Error converting data to data frame: ", e$message)
    return(NULL)
  })

  if (is.null(data_filtered)) {
    return(NULL)  # Skip file if data cannot be converted to dataframe
  }

  # Assigning column names
  colnames(data_filtered) <- unlist(strsplit(data$data$data$header, ","))

  # Filtering by ELEMENT and VTYPE
  data_filtered <- data_filtered %>%
    filter(ELEMENT == element, VTYPE == vtype)

  # Converting DT to date and filtering for entire year 2022
  data_filtered$DT <- as.Date(data_filtered$DT)
  data_filtered <- data_filtered %>%
    filter(year(DT) == rok)

  # Adding station identifier
  data_filtered$stanice_id <- stanice_id

  return(data_filtered)
}

# Function for getting data for all stations for year 2022
ziskat_data_ze_vsech_stanic <- function(stanice_s_informacemi, element, vtype, rok) {
  # Initializing empty dataframe
  vysledna_data <- data.frame()
```

```
  for (i in 1:nrow(stanice_s_informacemi)) {
    stanice_id <- stanice_s_informacemi$WSI[i]
    message("Processing station with WSI: ", stanice_id)
    data <- ziskat_data_stanice_pro_rok(stanice_id, element, vtype, rok)
    if (!is.null(data)) {
      # Adding data for each station
      vysledna_data <- rbind(vysledna_data, data)
    }
  }
  return(vysledna_data)
}
```

Now we set the variables: coordinates, monitored variable and its type, maximum distance, and year. In this case, we set the distance to 750 km to get all stations that measured temperature in 2022.

```
# Setting coordinates, year of interest, ELEMENT, VTYPE, maximum distance and specific date
lat <- 49.0679850   # Latitude
lon <- 13.5678442   # Longitude
element <- "T"      # ELEMENT parameter, e.g., "T" (temperature)
vtype <- "AVG"      # VTYPE parameter, e.g., "AVG" (average value)
max_vzdalenost_km <- 750  # Maximum distance in kilometers
rok <- 2022

# Getting list of stations that have available data for given year and variable
stanice_s_promennou <- ziskat_stanice_s_dostupnou_promennou(rok, element)

# Joining information from meta1 with filtering based on year and available coordinates
stanice_s_informacemi <- pripojit_meta1(stanice_s_promennou, rok)
stanice_s_informacemi <- stanice_s_informacemi[!is.na(stanice_s_informacemi$GEOGR1) &
                                                 !is.na(stanice_s_informacemi$GEOGR2), ]

# Filtering stations based on distance from given point
stanice_s_informacemi <- filtrovat_stanice_dle_vzdalenosti(stanice_s_informacemi,
                                                           lat, lon, max_vzdalenost_km)

# Save list of stations that meet conditions including their metadata
saveRDS(stanice_s_informacemi, file = "./data/stanice_s_informacemi.rds")
```

Now we just need to download data for selected stations and save them to the chmu_data_2022 table.

```
# Getting temperature data for all stations for entire year 2022
chmu_data_2022 <- ziskat_data_ze_vsech_stanic(stanice_s_informacemi, element, vtype, rok)

# Saving data to file in RDS format for further processing
saveRDS(chmu_data_2022, file = "./data/chmu_data_2022.rds")
```

3.4. Comparison with Data Worldwide

Daily data from various stations around the world can be obtained using the GSODR library. The data comes from GSOD, or Global Surface Summary of the Day. The data is provided by the National Centers for Environmental Information USA (NCEI) and is a valuable source of meteorological data with worldwide coverage. Currently, 40 stations in the Czech Republic are included. In addition to the availability of international data, the advantage is also the speed of processing without the need to download a large number of JSON files. This way, data from approximately 9,000 stations is available, some of which have data from 1929. The global coverage of stations can be seen in **Figure 32**.

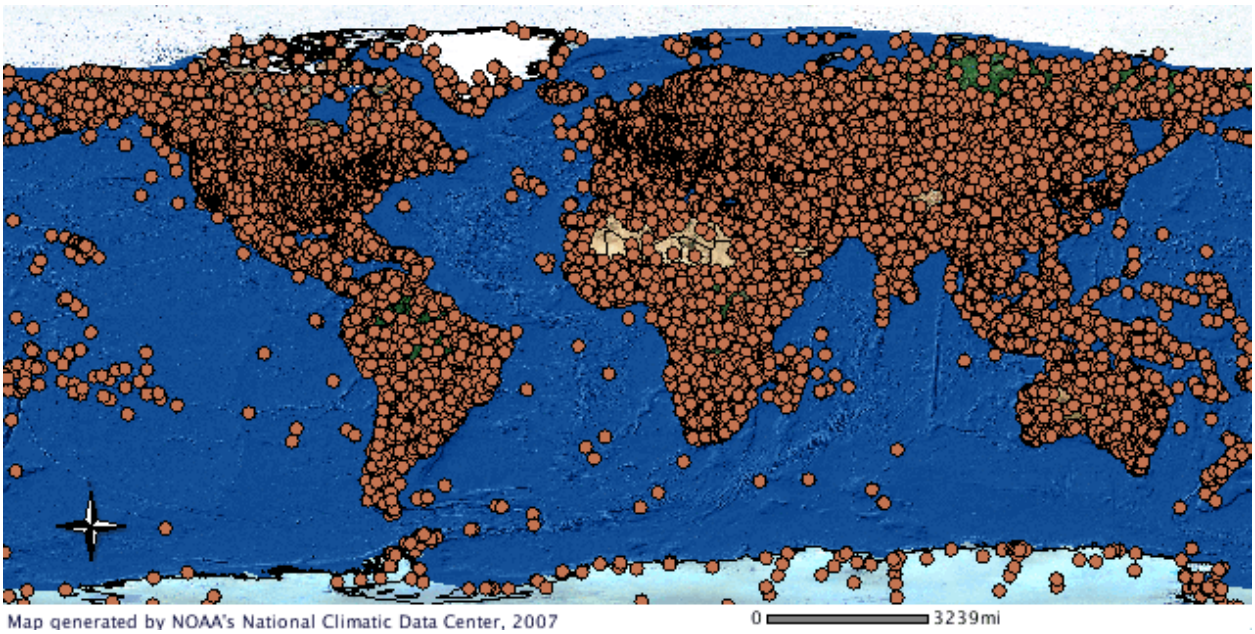

Figure 33: Map of stations in the GSOD database.

Methods

Let's try again to download data from the Churáňov station, whose ID in the GSOD database is "114570-99999". We'll use this to obtain daily data from 2022. Again, we'll load data from our two stations and join them with GSOD data.

```
# Find nearest stations within given radius - WGS84 coordinates and radius in kilometers
GSODR::nearest_stations(LAT = 49.0679850, LON = 13.5678442, distance = 20)

# Download data for given location and period
station_id <- "114570-99999"  # Churáňov
start_year <- 2022
end_year <- 2022

# Error handling for download attempt - GSOD is occasionally unavailable
tryCatch({
  # Try to download data from GSODR
  climate_data_gsod <- GSODR::get_GSOD(start_year:end_year, station = station_id)
}, error = function(e) {
  # In case of error, load prepared data from file
```

```
  message("Failed to download data, loading saved data from file.")
  climate_data_gsod <- readRDS("./data/gsod_Churanov_2022.rds")
})

# Loading saved TMS stations (see chapter 3.2)
df_wide.tms<-readRDS("./data/dve_tms_sumava.rds")

# Combining datasets into one with added 'locality' column
combined_data <- rbind(
  data.frame(Date = as.Date(climate_data_gsod$YEARMODA),
             Temp = climate_data_gsod$TEMP, Locality = "GSOD Churáňov"),
  data.frame(Date = as.Date(df_wide.tms[df_wide.tms$locality_id==
                                        "NPS_4210_D_T1",]$datetime),
             Temp = df_wide.tms[df_wide.tms$locality_id=="NPS_4210_D_T1",]$Thermo_T_mean,
             Locality = "NPS_4210_D_T1"),
  data.frame(Date = as.Date(df_wide.tms[df_wide.tms$locality_id==
                                        "4220_TMS",]$datetime),
             Temp = df_wide.tms[df_wide.tms$locality_id==
                                "4220_TMS",]$Thermo_T_mean,
             Locality = "4220_TMS")
)
```

Results

Now let's look at the average temperature progression in March 2022.

```
# Define data range for selection
start_date <- as.Date("2022-03-01")
end_date <- as.Date("2022-03-31")
filtered_data <- combined_data[combined_data$Date >= start_date
                               & combined_data$Date <= end_date,]

# Creating graph for selection
ggplot(filtered_data, aes(x = Date, y = Temp, color = Locality)) +
  geom_line() +
  scale_color_manual(values = c("#E69F00", "#000000", "#009E73")) +
  labs(
    title = "Average daily temperature from three sources",
    x = "Date",
    y = "Temperature (°C)",
    color = "Location"
  ) +  # Legend label
  theme_minimal()
```

Alternatively, we can download data for the entire country and filter for a selected day as in section 3.3.

```
# Enter date of interest
date_of_interest <- as.Date("2022-09-01")

# Download data for year selected by date of interest for Czech Republic
year_of_interest <- format(date_of_interest, "%Y")
climate_data_cz <- GSODR::get_GSOD(as.integer(year_of_interest),
```

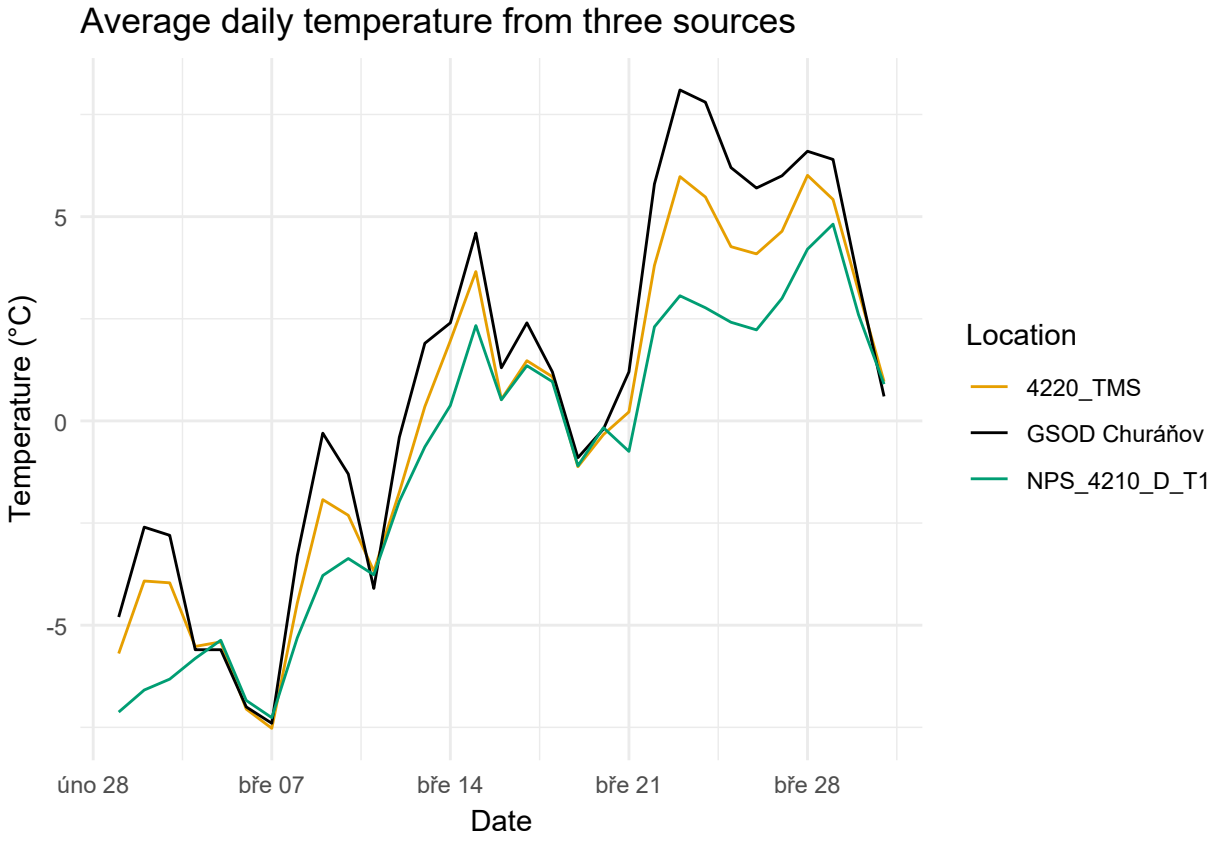

Figure 34: Here we can see that data in the GSOD database is not identical to CHMI data. This is partly due to a different method of calculating the average, and partly it may be due to further modification of data within GSOD.

```
                              station = NULL,
                              country = "CZE",
                              max_missing = NULL,
                              agroclimatology = FALSE)

# Filter data for specific date
filtered_data <- subset(climate_data_cz, YEARMODA == date_of_interest)
```

However, it will be more interesting to compare data from TMS stations with data from stations within 250 km, instead of the whole Czech Republic, since the location is close to the borders.

```
# Period
start_year <- 2022
end_year <- 2022

# Find nearest stations within given radius (in km)
stations_nearby <- GSODR::nearest_stations(LAT = 49.0679850, LON = 13.5678442,
                                           distance = 250)

# It's evident there are errors in station data, we'll remove obvious ones
stations_nearby<-stations_nearby[!stations_nearby$COUNTRY_NAME=="GUAM",]
stations_nearby<-stations_nearby[!stations_nearby$COUNTRY_NAME=="IRAN",]
stations_nearby<-stations_nearby[!stations_nearby$COUNTRY_NAME=="IRAQ",]
stations_nearby<-stations_nearby[!is.na(stations_nearby$`ELEV(M)`),]

# Download GSOD data for all stations within 250 km
# Using lapply function to download data from multiple stations
climate_data_nearby <- lapply(stations_nearby$STNID, function(station_id) {
  get_GSOD(years = start_year:end_year, station = station_id)
})

# Merging all downloaded data into one data frame
climate_data_nearby <- do.call(rbind, climate_data_nearby)

# For faster processing we'll save the data, this part of code doesn't run automatically
# to avoid overloading storage, if you change the radius, run again.
# Current setting downloads 291 stations.

saveRDS(climate_data_nearby, file = "./data/gsod_nearby_2022.rds")
```

And now we'll create a graph again for the selected day (September 1, 2022), where we'll add a simple regression of temperature and elevation with confidence intervals. This time, however, data from surrounding stations within GSOD is used.

```
# Loading saved TMS stations (see chapter 3.2)
df_wide.tms<-readRDS("./data/dve_tms_sumava.rds")
climate_data_nearby<-readRDS("./data/gsod_nearby_2022.rds")
# Adding elevation
df_wide.tms$elevation <- 1120

# Enter date of interest
date_of_interest <- as.Date("2022-09-01")
```

```r
# Filtering data for specific date
filtered_data <- subset(climate_data_nearby, as.Date(YEARMODA) == date_of_interest)
filtered_data_tms <- subset(df_wide.tms, as.Date(datetime) == date_of_interest)

# Check if data is available for given date
if (nrow(filtered_data) > 0) {
  # Create scatter plot temperature vs. elevation with trend line
  p <- ggplot(filtered_data, aes(x = ELEVATION, y = TEMP)) +
    geom_point(color = "blue", alpha = 0.7) +
    # Adding regression line and confidence intervals
    geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed",
                fill = "lightblue", alpha = 0.3) +
    labs(title = paste("Temperature vs. elevation on", date_of_interest, "source: GSOD"),
         x = "Elevation (m a.s.l.)", y = "Average daily temperature (°C)") +
    theme_minimal()
  # Adding data from logger stations as red points
  p <- p + geom_point(data = filtered_data_tms, aes(x = elevation, y = Thermo_T_mean),
            color = "red", size = 3)
  print(p)
} else {
  cat("No data available for selected date:", date_of_interest, "\n")
}
```

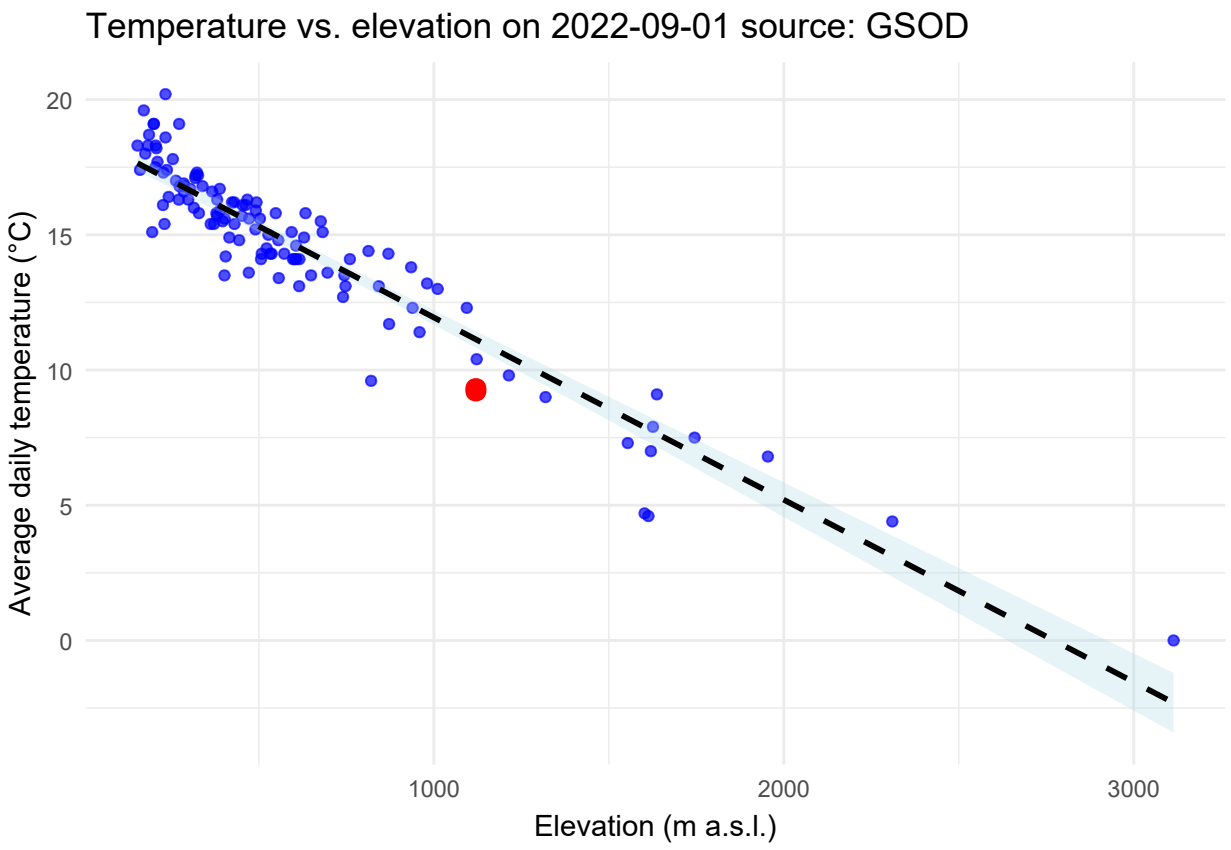

Figure 35: Although the graph looks similar, from the distribution of stations along the elevation gradient it is evident that these are different stations.

4. Working with Rasters

Currently, many climatic data are available in raster form, from global to purely local. They differ not only in resolution (pixel can have an edge in the order of meters or kilometers) but also in the time range of data and type of variables. For many questions, we can thus obtain relevant data without having to install climate stations. However, it is always necessary to assess whether the given data is suitable for solving the given question, primarily with regard to resolution, but also the data source. In this chapter, we will show several examples of rasters and how to work with them in the R environment. For rasters, it is often simpler to use a geographic information system (GIS), such as QGIS.

4.1. Microclimate Rasters for Šumava NP Processed by IBOT

For Šumava National Park, we prepared a set of microclimatic maps covering the territory of Šumava NP and Bavarian Forest NP in 2021. The data is publicly published (Brůna et al., 2023) and can be worked with directly in the R environment. The maps were created using data from a network of microclimatic stations from October 2019 - October 2020. The effect of vegetation cover and topography was incorporated using laser scanning (LiDAR) data. The interpolation models take into account not only topographic influences on microclimate but also the influence of density and type of vegetation cover. The maps are suitable for solving detailed questions concerning climate in forest stands and adjacent open areas at 3 different heights - below ground, 15 cm above ground, and 200 cm above ground.

Preparation of required R environment libraries.

The following are available in the **Zenodo** repository (filename can be used in the following code):

Soil temperature (-8 cm)

- Average temperature = mean temperature *T.soil_8_cm.mean.tif*

Air temperature at ground level (15 cm)

- Average temperature *T.air_15_cm.mean.tif*
- Maximum temperature = 95th percentile of daily maximum temperatures *T.air_15_cm.max.95p.tif*
- Minimum temperature = 5th percentile of daily minimum temperatures *T.air_15_cm.min.5p.tif*

Air temperature (200 cm)

- Average temperature *T.air_200_cm.mean.tif*
- Maximum temperature = 95th percentile of daily maximum temperatures *T.air_200_cm.max.95p.tif*
- Minimum temperature = 5th percentile of daily minimum temperatures *T.air_200_cm.min.5p.tif*
- GDD = sum of growing degree days above base temperature (base 5°C) *T.air_200_cm.GDD5.tif*

Using the following code, we will download the map of average temperature at 15 cm above ground and load it into the **zeta** object.

```r
# URL for file download (file T.air_15_cm.mean.tif from Zenodo archive, size 97.0 MB)
soubor <- c("T.air_15_cm.mean.tif")

url <- paste0("https://zenodo.org/record/6352641/files/",soubor)

# Path where to save file
destfile <- paste0("./data/rastry/", soubor)

# Create folder if it doesn't exist
dir.create("./data/rastry", recursive = TRUE, showWarnings = FALSE)

# download raster using curl
curl::multi_download(url, destfile, resume = T)

# Loading raster using terra library
zeta <- terra::rast(destfile)
```

Map

After downloading, we'll visualize the entire map using a basic plot.

```r
# Setting space for legend
par(mar = c(5, 4, 4, 5))  # Adding space on right side for legend
par(oma = c(2, 1, 1, 1))  # Increasing outer bottom margin

# Basic plot of raster using plot() function
plot(zeta, col = viridis::viridis(100),
     xlab = "Longitude", ylab = "Latitude")

# Adding label to legend with unit (°C)
mtext("Temperature (°C)", side = 4, line = 3)  # Adding label on right side
```

Histogram

Now let's look at the distribution of temperatures using a histogram, which will show the frequency of different values.

```r
# Calculate average value of raster
values_zeta <- values(zeta)
mean_zeta <- mean(values_zeta, na.rm = TRUE)

# Create histogram for raster values
hist_zeta <- hist(values_zeta, main = "Temperature Histogram",
                  xlab = "Average Annual Temperature (°C)", ylab = "Frequency",
                  col = "lightblue", breaks = 20,
                  probability = TRUE, border = "gray")

# Add vertical line indicating average
abline(v = mean_zeta, col = "red", lwd = 2)

# Add label with average value
```

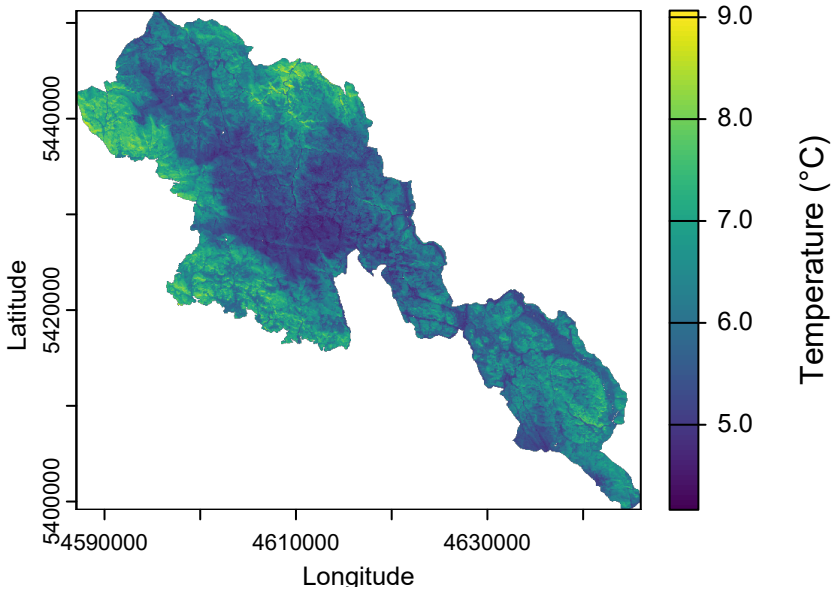

Figure 36: Map of annual average air temperature 15 cm above ground in Šumava NP (10-2019 - 10-2020). The map shows a strong influence of elevation on temperature. The influence of topography is also apparent.

```
text(mean_zeta, par("usr")[4] * 0.92,
     labels = paste0("Average: ", round(mean_zeta, 2)," °C"),
     col = "red", pos = 4)
```

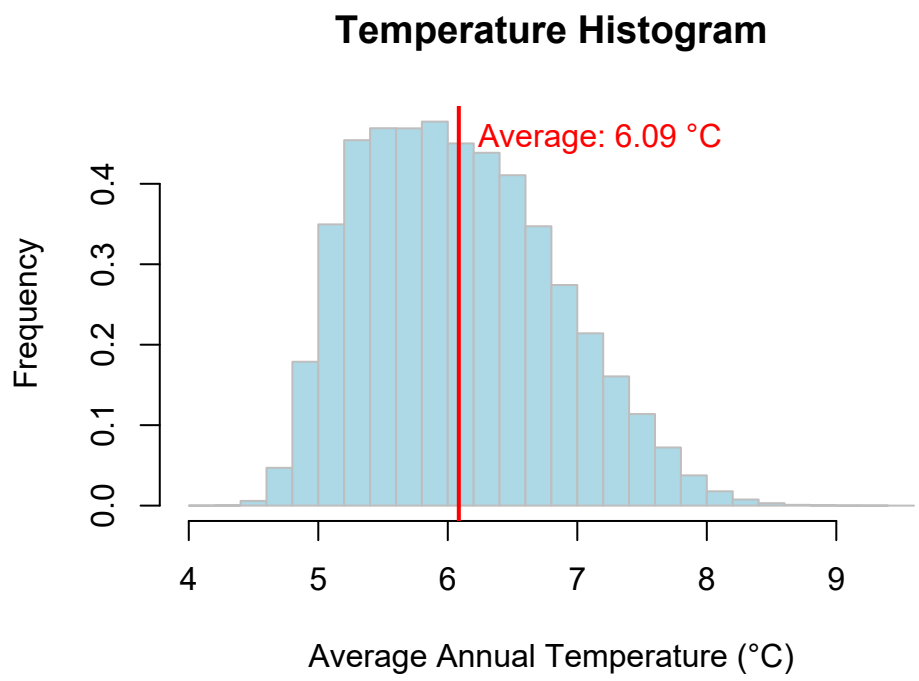

Figure 37: The histogram shows that the annual average is 6.09°C, but in different areas it can range from 4.4°C to 8.6°C.

Subset

Now let's look at a smaller area to utilize the high resolution of the maps. For example, we'll crop the map to a 1000 x 1000 m square around Březnická Lodge.

```
# Define center coordinates in WGS84 (EPSG:4326)
lat <- 48.9712522 # Březnická Lodge
lon <- 13.4825903
coords <- data.frame(lon, lat)

# Create sf object with coordinates in EPSG:4326
point_wgs84 <- st_as_sf(coords, coords = c("lon", "lat"), crs = 4326)

# Transform point to coordinate system matching raster data
# (3-degree Gauss-Kruger zone 4, EPSG:31468)
point_epsg31468 <- st_transform(point_wgs84, 31468)

# Get coordinates for creating rectangle (bounding box) around point
point_coords <- st_coordinates(point_epsg31468)
```

```r
x_center <- point_coords[1, "X"]
y_center <- point_coords[1, "Y"]

# Define subset size (e.g., 500x500 meters around point)
buffer_size <- 500  # half size (in meters)
xmin <- x_center - buffer_size
xmax <- x_center + buffer_size
ymin <- y_center - buffer_size
ymax <- y_center + buffer_size

# Create rectangle (bbox)
bbox <- terra::ext(xmin, xmax, ymin, ymax)

# Crop raster using rectangle
zeta_cropped <- crop(zeta, bbox)

# Set margins on map edges for legend
par(mar = c(4, 4, 4, 5))  # Add space on right side for legend
par(oma = c(2, 1, 1, 1))  # Increase outer bottom margin

# Basic plot of raster using plot() function
plot(zeta_cropped, col = viridis::viridis(100),
     xlab = "Longitude (m), EPSG:31468",
     ylab = "Latitude (m)")

# Add label to legend with unit (°C)
mtext("Temperature (°C)", side = 4, line = 3)  # Add label on right side

# Add point with coordinates on map ('x' symbol)
points(x_center, y_center, pch = 4, col = "red", cex = 1.5)  # pch=4 is 'x' symbol
```

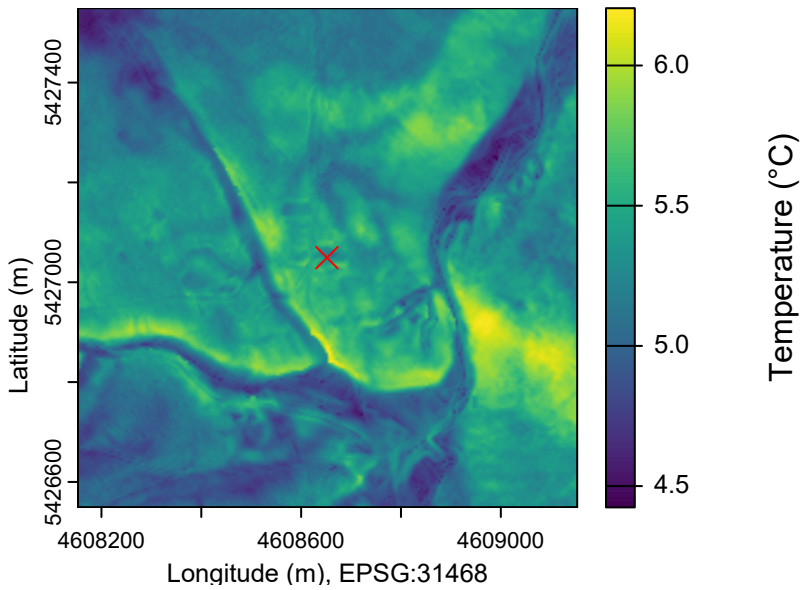

Figure 38: Subset: this map shows warmer, south-facing meadow areas below Březnická Lodge (marked with x) and colder valley positions. Again, this is annual average temperature (10-2019 - 10-2020).

Extracting Values from Rasters for Points

Values in raster form are good for creating maps, but more often we need values for points of interest as input for analysis. The following example first converts points to the map's coordinate system, then extracts values for the points and adds them to the table.

```
# Loading lat/lon coordinates from CSV that contains 'lat' and 'lon' columns
coords_data <- read.csv("data/rastry/souradnice.csv", sep=";")

# Creating sf object from coordinates in EPSG:4326 (WGS84)
points_wgs84 <- st_as_sf(coords_data, coords = c("lon", "lat"), crs = 4326)

# Transforming points to EPSG:31468
points_epsg31468 <- st_transform(points_wgs84, 31468)

# Converting sf object to format suitable for terra::extract
coords_matrix <- st_coordinates(points_epsg31468)

# Extracting values from raster for coordinates using bilinear interpolation
values <- terra::extract(zeta, coords_matrix, method = "bilinear")

# Joining extracted values with coordinates and saving to new table
final_data <- cbind(coords_data, values)
default_kable(final_data)
```

Table 9: The last point has NA value in the temperature column, likely because it was outside the map area. We can easily verify this by plotting the points on the map.

id	lat	lon	altitude	T.air_15_cm.mean
1	49.11978	13.30771	1140.50	5.333770
2	49.07077	13.38393	1132.40	5.249397
3	48.98027	13.63936	1015.00	5.057910
4	49.05457	13.49517	907.00	6.836538
5	48.98433	13.47457	1207.10	4.736646
6	48.99739	13.52126	1136.30	4.898316
7	48.92075	13.66820	902.20	4.993112
8	48.87504	13.76450	1055.60	5.354843
9	48.96517	13.49173	1176.43	5.175743
10	48.98573	13.79490	1247.00	NaN

We can also visualize the points directly on the map with the raster. This also helps us verify the correctness of the points.

```
# Setting space for legend and plot
par(mar = c(4, 4, 4, 5))  # Adding space on right side for legend
par(oma = c(2, 1, 1, 1))  # Increasing outer bottom margin

# Plotting raster - for checking that all points are inside map
plot(zeta, main = "Visualization of raster with coordinates",
     col = viridis(100),  # Using viridis color scale
     xlab = "Longitude (m), EPSG:31468",
     ylab = "Latitude (m)")

# Adding legend with unit (°C)
mtext("Temperature (°C)", side = 4, line = 3)

# Adding all points from coordinates as crosses (pch=4 is 'x' symbol)
points(coords_matrix[,1], coords_matrix[,2], pch = 4, col = "red", cex = 1.5)
```

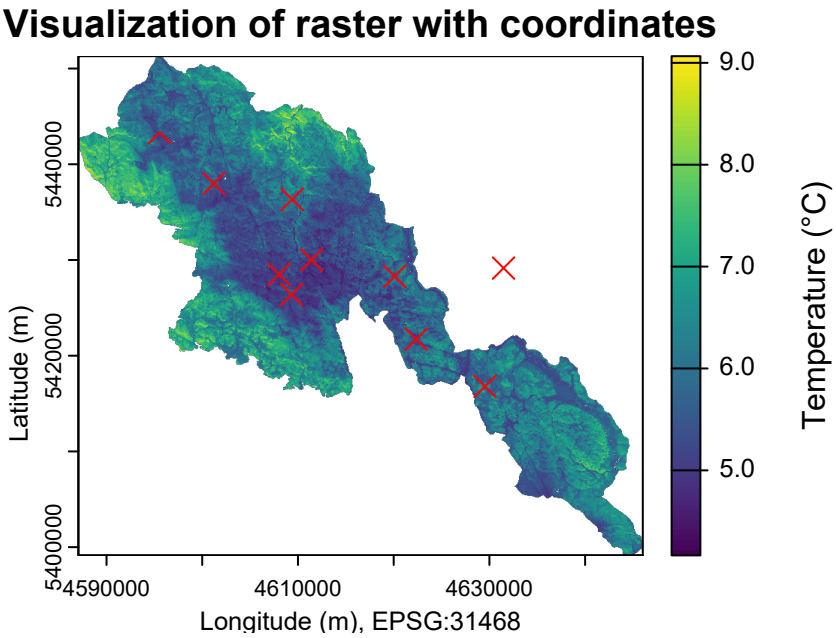

Figure 39: Here we can see that one of the points was indeed outside the map area.

4.2. External Sources

ForestClim

Now let's look at the external source ForestClim (Haesen et al., 2023). This is a set of bioclimatological variable maps covering European forests with 25 m resolution. The map set is based on measurements by microclimatic dataloggers, predominantly TOMST TMS-4. Its use therefore targets temperature differences in vegetation stands. The high resolution allows comparison of even very close locations. In this case, bioclimatological variables are available that summarize microclimate using 11 derived variables:

ForestClim_01 = mean annual temperature

ForestClim_02 = mean diurnal temperature range

ForestClim_03 = isothermality

ForestClim_04 = temperature seasonality

ForestClim_05 = maximum temperature of warmest month

ForestClim_06 = minimum temperature of coldest month

ForestClim_07 = annual temperature range

ForestClim_08 = mean temperature of wettest quarter

ForestClim_09 = mean temperature of driest quarter

ForestClim_10 = mean temperature of warmest quarter

ForestClim_11 = mean temperature of coldest quarter

Individual files are around 6 GB in size, so it's easier to download the data beforehand here: https://figshare.com/articles/dataset/ForestClim_Bioclimatic_variables_for_microclimate_temperatures_of_European_forests/22059125?file=39164684

For our example, we'll use mean annual temperature, i.e., layer ForestClim_01. We have saved a subset for the Šumava NP area in the data folder. The following example will again generate a simple map.

```
# Loading ForestClim raster using terra
forest_raster <- terra::rast("./data/rastry/ForestClim_01_CZE_NPS.tif")

# Displaying new raster
par(mar = c(5, 4, 4, 5))  # Setting margins
plot(forest_raster, main = paste0("Visualization of raster ", "ForestClim_01"),
     col = viridis::viridis(100),
     xlab = "Longitude", ylab = "Latitude")

# Adding label to legend
mtext("Temperature (°C)", side = 4, line = 3)  # Adding label on right side
```

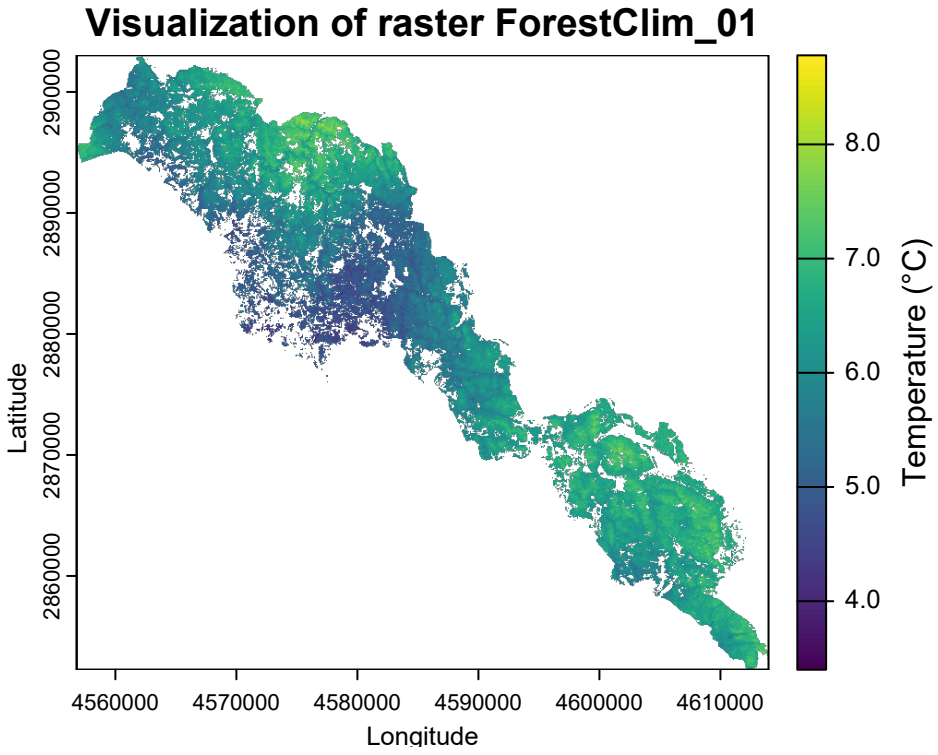

Figure 40: Visualization of ForestClim_01 raster subset for Šumava NP area

Let's look at the histogram of mean annual temperatures again.

```
# Histogram for ForestClim raster
forest_mean_value <- mean(values(forest_raster), na.rm = TRUE)

# Histogram for ForestClim
hist_forest <- hist(values(forest_raster), main = "ForestClim Histogram",
                    xlab = "Mean Annual Temperature (°C)", ylab = "Count",
```

```
                    col = "lightgreen", breaks = 20,
                    probability = TRUE, border = "gray")

# Adding vertical line indicating average for ForestClim raster
abline(v = forest_mean_value, col = "darkred", lwd = 2)

# Adding label for mean value
text(forest_mean_value, par("usr")[4] * 0.9, labels =
     paste0("Mean: ", round(forest_mean_value, 2),"°C"),
     col = "darkred", pos = 4)
```

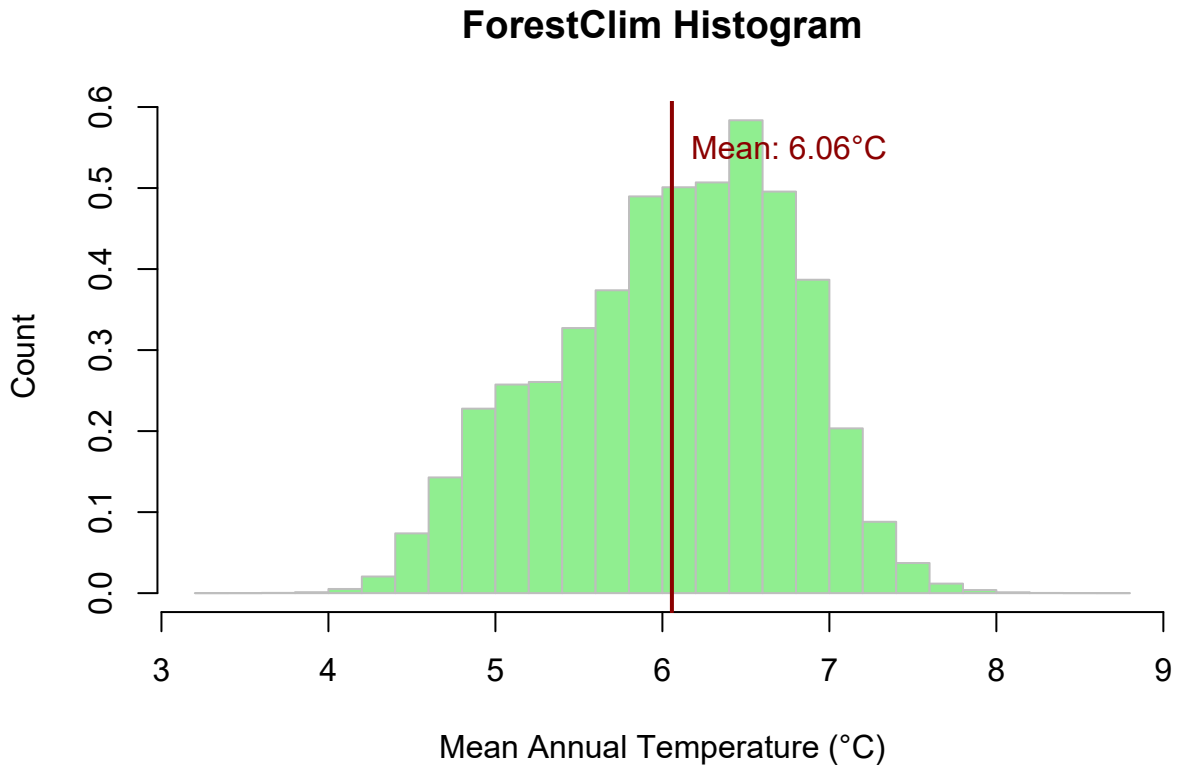

Figure 41: Although the average is very similar, the counts of values in this raster are different.

And again let's look at the detail of the map around Březnická Lodge

```
# Using same point_wgs84, defined in previous example (Březnická Lodge)
# Transform point to EPSG:3035 (European Lambert Conformal Conic)
point_epsg3035 <- st_transform(point_wgs84, 3035)

# Get coordinates for creating rectangle (bounding box) around point
point_coords <- st_coordinates(point_epsg3035)
x_center <- point_coords[1, "X"]
y_center <- point_coords[1, "Y"]

# Define subset size (e.g., 500 meters around point)
```

```r
buffer_size <- 500
xmin <- x_center - buffer_size
xmax <- x_center + buffer_size
ymin <- y_center - buffer_size
ymax <- y_center + buffer_size

# Create rectangle (bbox)
bbox <- terra::ext(xmin, xmax, ymin, ymax)

# Crop raster using rectangle (for ForestClim raster)
forest_raster_cropped <- crop(forest_raster, bbox)

# Setting space for legend
par(mar = c(4, 4, 4, 5))  # Adding space on right side for legend
par(oma = c(2, 1, 1, 1))  # Increasing outer bottom margin

# Basic plot of raster using plot() function
plot(forest_raster_cropped, main = paste0("Visualization of ForestClim raster subset"),
     col = viridis::viridis(100),
     xlab = "Longitude (m), EPSG:3035",
     ylab = "Latitude (m)")

# Adding label to legend
mtext("Raster values", side = 4, line = 3)  # Label on right side

# Adding point with coordinates on map ('x' symbol)
points(x_center, y_center, pch = 4, col = "red", cex = 1.5)  # pch=4 is 'x' symbol
```

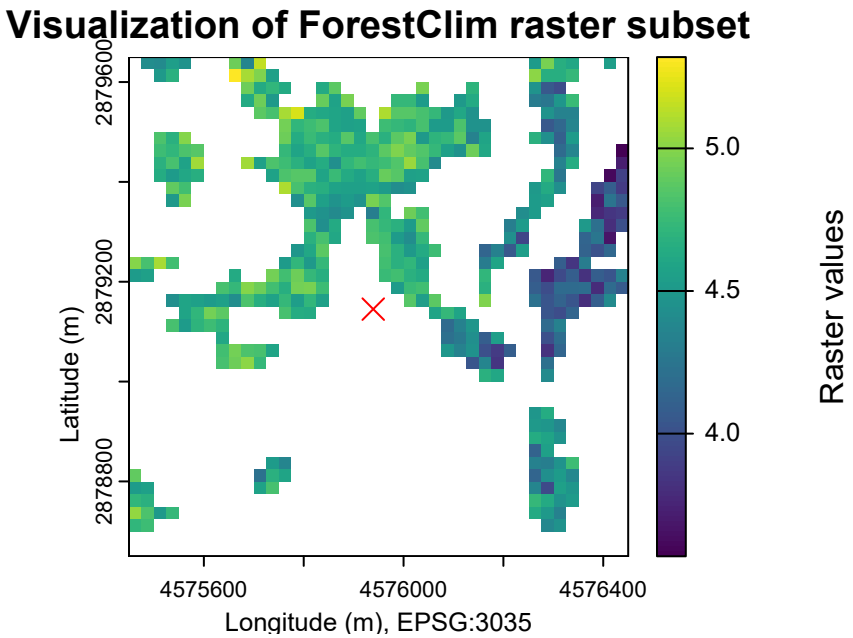

Figure 42: Here we can see a weakness of this dataset. Besides lower spatial resolution (larger pixels), it's the coverage of only forest stands, which are defined by Copernicus layers (more in Haesen et al., 2023), and thus disturbed areas, regeneration, and lower stands are not included.

Comparison of Histograms between ForestClim and IBOT Rasters

It makes sense to compare with the average temperature raster created by the Institute of Botany. This part relies on variables that you created when running the code in section 4.1.

```
# Values for both rasters
values_forest <- values(forest_raster)

# Calculation of common histogram breaks
breaks <- seq(min(c(values_zeta, values_forest), na.rm = TRUE),
              max(c(values_zeta, values_forest), na.rm = TRUE), length.out = 30)

# Plotting first histogram for T.air_15_cm.mean.tif (in memory from previous calculations)
plot(hist_zeta, col = rgb(0, 0, 1, 0.5), xlim = range(breaks),
     ylim = c(0, max(hist_zeta$counts)),
     main = "Combined histogram of rasters with dual Y axis",
     xlab = "Value", ylab = soubor,
     cex.main = 1.2)  # Reducing title size

# Adding line and label for T.air_15_cm.mean.tif mean (average above line)
abline(v = mean_zeta, col = "blue", lwd = 2)
text(mean_zeta, max(hist_zeta$counts) * 0.75, labels =
       paste0("Mean: ", round(mean_zeta, 2),"°C"), col = "blue", pos = 4)

# Plotting second y axis on right side
par(new = TRUE)
plot(hist_forest, col = rgb(0, 1, 0, 0.5), xlim = range(breaks),
     ylim = c(0, max(hist_forest$counts)),
     axes = FALSE, xlab = "", ylab = "", main="")
axis(side = 4)
mtext("ForestClim Frequency", side = 4, line = 3)

# Adding line and label for ForestClim mean (average below line)
abline(v = forest_mean_value, col = "darkgreen", lwd = 2)
text(forest_mean_value, max(hist_forest$counts) * 0.5, labels =
       paste0("Mean: ", round(forest_mean_value, 2),"°C"), col = "darkgreen", pos = 4)

# Adding legend
legend("topright", legend = c("T.air_15_cm.mean", "ForestClim 01"),
       fill = c(rgb(0, 0, 1, 0.5), rgb(0, 1, 0, 0.5)))
```

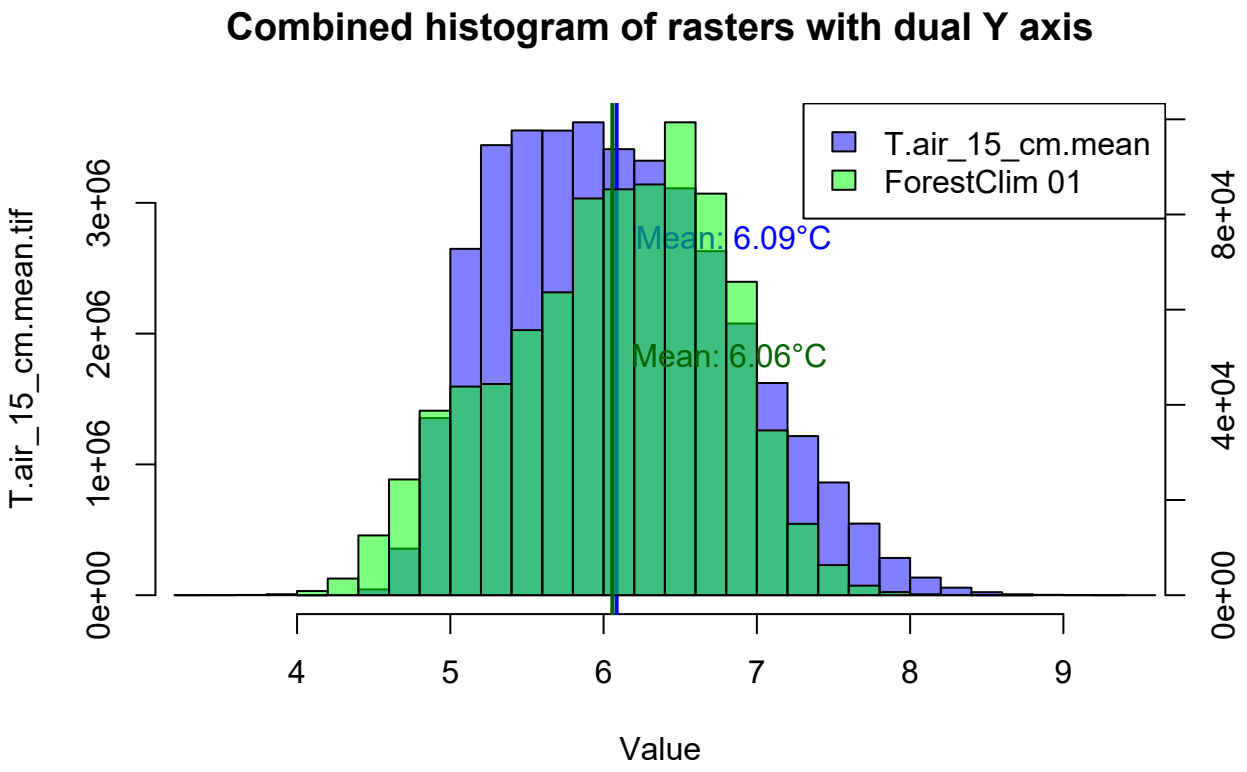

Figure 43: Although the average of both datasets doesn't differ much, the histogram shows that the microclimatic map processed by the Institute of Botany has more colder values than average and also locations with higher temperatures. Both probably relate to ForestClim's forest-only crop, however, it may also indicate a greater influence of stands on microclimate in the dataset created by the Institute of Botany.

Downscaled E-OBS Data Using the *easyclimate* Library

The easyclimate library provides access to daily climate data with 1 km resolution for Europe (precipitation, minimum and maximum temperatures) from the European climate database hosted at the University of Natural Resources and Life Sciences, Vienna, Austria. Data is currently available from 1950 to 2022.

The input climate dataset is based on downscaled E-OBS (European Observations) rasters. It was originally created by A. Moreno and H. Hasenauer - https://doi.org/10.1002/joc.4436 and further developed by W. Rammer, C. Pucher and M. Neumann. The current version is v4 - detailed description in this document - https://doi.org/10.6084/m9.figshare.22962671.v1). For a detailed description of the easyclimate library, read this article - https://doi.org/10.1016/j.envsoft.2023.105627 (open access version https://doi.org/10.32942/osf.io/mc8uj) or visit the library website - https://verughub.github.io/easyclimate/.

The input E-OBS rasters are based on interpolation of data from 7,852 climate stations in Europe. Thanks to this, they are suitable for characterizing macroclimate regardless of forest influence or other landscape cover. Through downscaling, the resolution was increased to 1 km, which still cannot capture microclimatic differences.

Example of downloading a raster for a selected area. The advantage is that we can download multiple days at once. We'll load the Šumava NP boundary in UTM 33N coordinate system, convert it to the data source coordinate system (WGS 84) and download data for May 1-3, 2020.

```
## Loading shapefile from specified path (EPSG:32633)
nps_region <- vect("./data/rastry/NPS_UTM33N.shp")

## Converting shapefile to required coordinate system (EPSG:4326 for lon lat coordinates)
nps_region <- project(nps_region, "EPSG:4326")

## Downloading Tmax values for Šumava NP area between May 1-3, 2020
nps_temp <- get_daily_climate(
  coords = nps_region,
  climatic_var = "Tmax", # can be changed to "Prcp" for precipitation or "Tmin" for min temperatures
  # mean temperature is not available here, but is commonly calculated as (Tmax+Tmin)/2
  period = "2020-05-01:2020-05-03",
  output = "raster"
)
```

The output **nps_temp** is a SpatRaster object with 3 layers (for each of the 3 days). Now let's display maps for individual days.

```
# Plotting individual layers and adding nps_region polygon boundaries
ggplot() +
  geom_spatraster(data = nps_temp) +
  facet_wrap(~lyr, ncol = 3) +
  scale_fill_distiller(palette = "RdYlBu", na.value = "transparent") +
  geom_spatvector(data = nps_region, fill = NA) +
  labs(fill = "Maximum\ntemperature \n(°C)") +
  scale_x_continuous(breaks = seq(13.3, 13.9, by = 0.3)) +
  scale_y_continuous(breaks = seq(48.8, 49.2, by = 0.2)) +
  theme_minimal()
```

Since data is available from 1950, we can use it well for comparison.

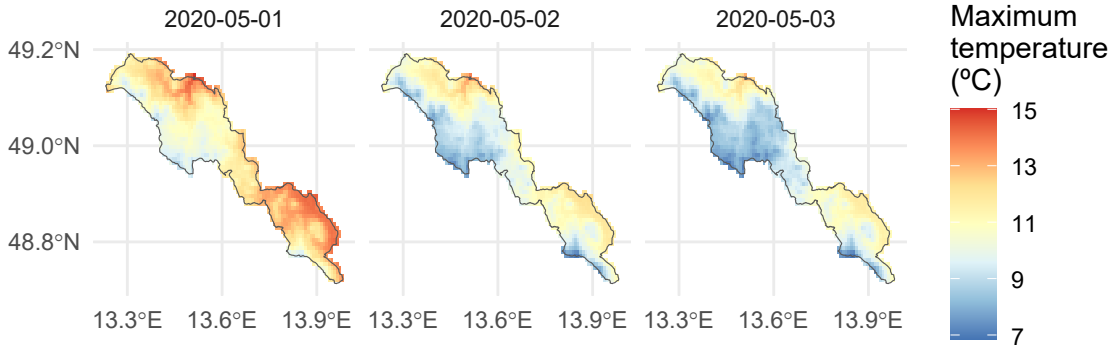

Figure 44: The three maps show maximum temperatures for individual days May 1-3, 2020.

```
## Downloading Tmax values for NPS area between May 1-3, 1950
nps_temp_1950 <- get_daily_climate(
  coords = nps_region,
  climatic_var = "Tmax",
  period = "1950-05-01:1950-05-03",
  output = "raster"
)

# Plotting individual layers (days) and adding nps_region polygon boundaries
ggplot() +
  geom_spatraster(data = nps_temp_1950) +
  facet_wrap(~lyr, ncol = 3) +
  scale_fill_distiller(palette = "RdYlBu", na.value = "transparent") +
  geom_spatvector(data = nps_region, fill = NA) +
  labs(fill = "Maximum\ntemperature \n(ºC)") +
  scale_x_continuous(breaks = seq(13.3, 13.9, by = 0.3)) +
  scale_y_continuous(breaks = seq(48.8, 49.2, by = 0.2)) +
  theme_minimal()
```

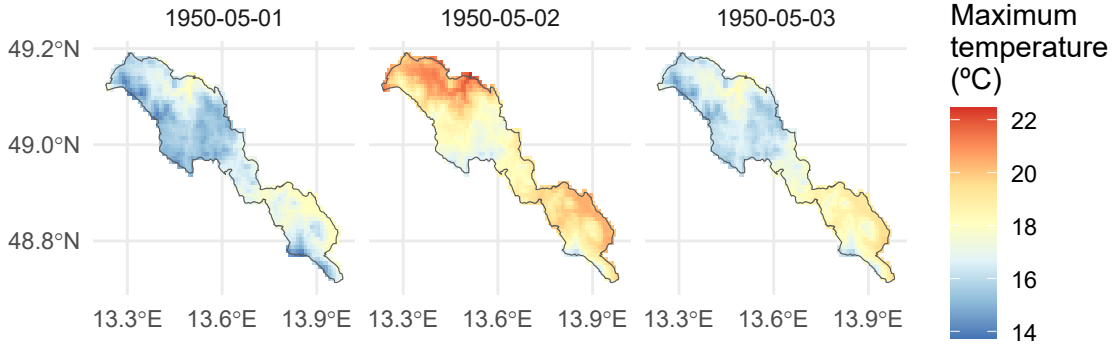

Figure 45: The three maps show maximum temperatures for individual days May 1-3, 1950. The legend has a different range of values.

Now we can calculate the difference for individual days between 2020 and 1950 and display in maps again. Daily temperature heterogeneity is high, so in analyses it's usually better to aggregate data over longer time periods - months, seasons, or years and compare those between each other.

```r
# Calculate difference between temperatures (nps_temp - nps_temp_1950)
nps_temp_diff <- nps_temp - nps_temp_1950

ggplot() +
  geom_spatraster(data = nps_temp_diff) +
  facet_wrap(~lyr, ncol = 3) +
  scale_fill_distiller(palette = "RdYlBu", na.value = "transparent") +
  geom_spatvector(data = nps_region, fill = NA) +
  labs(fill = "Difference \nmaximum \ntemperature \n2020-1950 \n(ºC)") +
  scale_x_continuous(breaks = seq(13.3, 13.9, by = 0.3)) +
  scale_y_continuous(breaks = seq(48.8, 49.2, by = 0.2)) +
  theme_minimal()
```

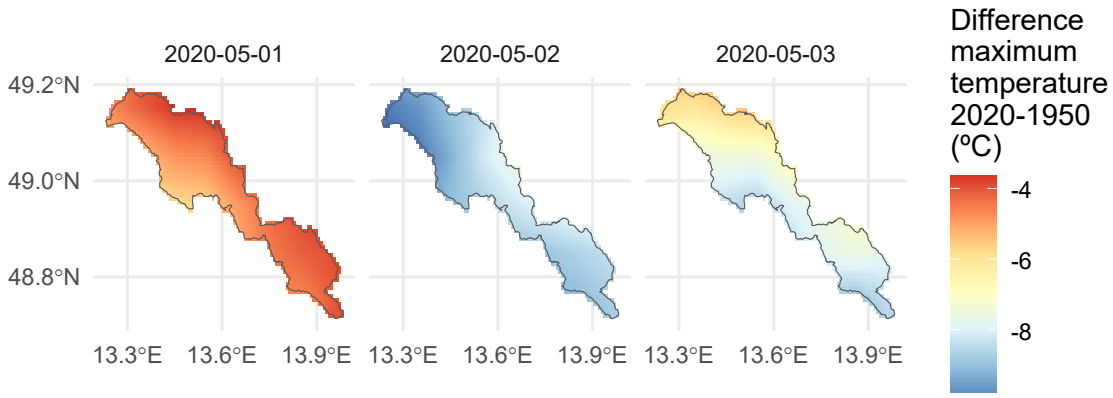

Figure 46: Differences between rasters between 2020 and 1950 were very different in individual days. When comparing individual days, it's mainly about specific weather and it's difficult to infer the influence of climate change.

Interactive Map Output

The following example demonstrates how to visualize a map in an interactive environment with a base map. This can be useful if you want to show results to someone who doesn't use R or GIS but wants to explore the data more than just from images.

```
if (is_html_output || is_interactive) { # Code for HTML version or interactive mode
  nps_temp_diff_1 <- nps_temp_diff[[1]]  # Select layer as needed
  nps_temp_diff_1_raster <- as(nps_temp_diff_1, "Raster")
  raster_values <- raster::values(nps_temp_diff_1_raster)

  # Color palette for visualization
  pal <- colorNumeric(palette = "RdYlBu", domain = raster_values,
                      na.color = "transparent")

  # Creating interactive map
  leaflet() %>%
    addTiles() %>%
    addRasterImage(nps_temp_diff_1_raster, colors = pal, opacity = 0.8) %>%
    addPolygons(data = as(nps_region, "Spatial"), fill = NA, color = "black") %>%
    addLegend(pal = pal, values = raster_values, title = "Difference Tmax 2020-1950 (ºC)")

} else { # Code for non-HTML version (PDF/Word - static image)
  nps_temp_diff_1 <- nps_temp_diff[[1]]  # Select layer as needed
  nps_temp_diff_1_raster <- as(nps_temp_diff_1, "Raster")
  raster_values <- raster::values(nps_temp_diff_1_raster)

  # Color palette
  pal <- colorNumeric(palette = "RdYlBu", domain = raster_values,
                      na.color = "transparent")

  # Creating interactive map that we'll save as HTML
  map <- leaflet() %>%
    addTiles() %>%
    addRasterImage(nps_temp_diff_1_raster, colors = pal, opacity = 0.8) %>%
    addPolygons(data = as(nps_region, "Spatial"), fill = NA, color = "black") %>%
    addLegend(pal = pal, values = raster_values, title = "Difference Tmax 2020-1950 (ºC)")

  # Saving widget as HTML file
  htmlwidgets::saveWidget(map, "./images/map.html", selfcontained = TRUE)

  # Creating static snapshot of map using webshot
  #webshot("./images/map.html", file = "./images/map.png", cliprect = "viewport")
  webshot2::webshot("./images/map.html", file = "./images/map.png", cliprect = "viewport",
                    vwidth = 1000, vheight = 800, delay = 5)


  # Inserting image into document
  knitr::include_graphics("./images/map.png", dpi=150)
}
```

If you use the *easyclimate* library, please cite both the relevant data source and the library: Cruz-Alonso V, Pucher C, Ratcliffe S, Ruiz-Benito P, Astigarraga J, Neumann M, Hasenauer H, Rodríguez-Sánchez F (2023). "The easyclimate R package: Easy access to high-resolution daily climate data for Europe." *Environmental*

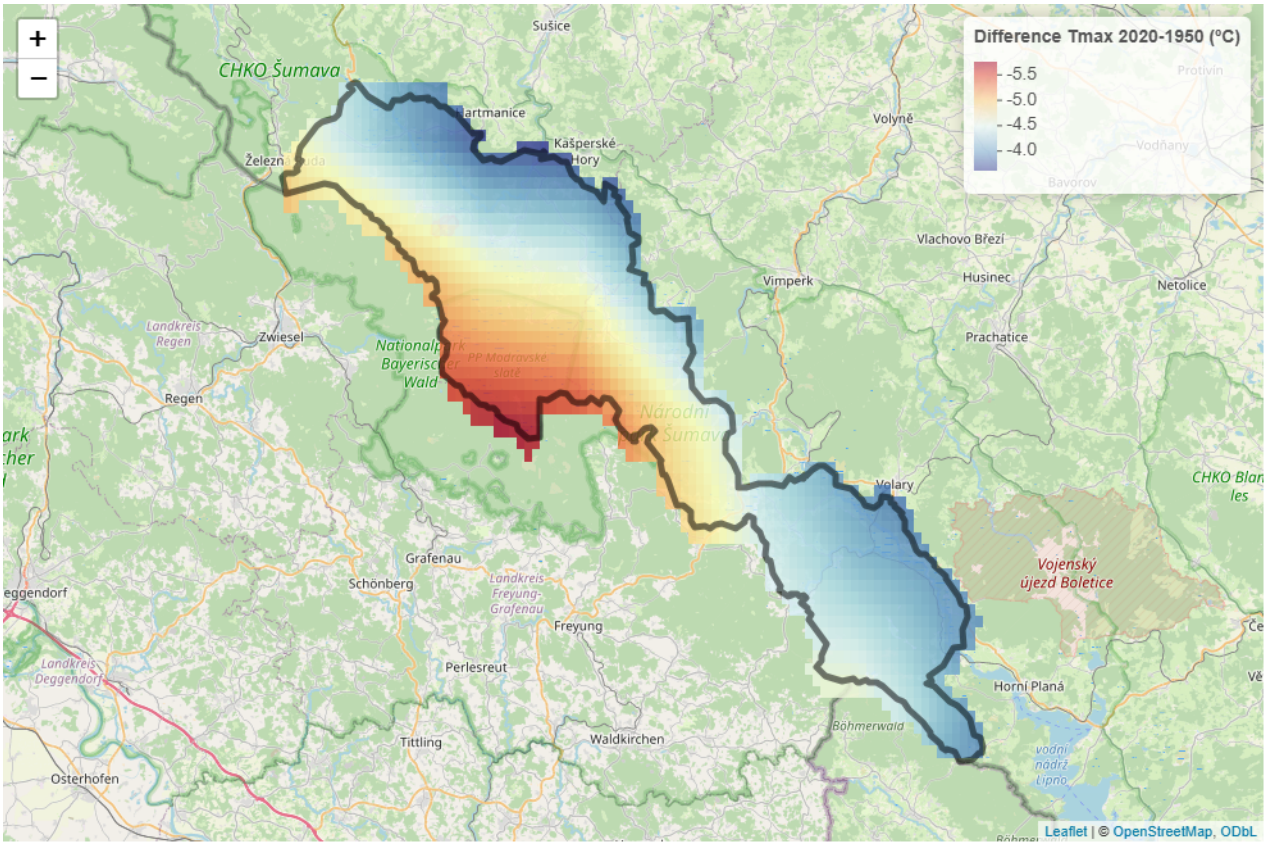

Figure 47: Interactive map uses online base map. However, the main advantage is the ability to easily manipulate the map, which cannot be shown in PDF format.

Modelling & Software, 105627. https://doi.org/10.1016/j.envsoft.2023.105627.

For further tutorials, visit the articles on the library website: https://verughub.github.io/easyclimate/.

4.3. Microclimate Rasters for České Švýcarsko National Park Processed by IBOT

Similar to Šumava NP, a microclimatic atlas was also created for České Švýcarsko NP and Saxon Switzerland NP. Using the following code, you can again download a selected map and visualize it using a map in the R environment.

https://gitlab.ibot.cas.cz/matej.man/microclimate-atlas-public/

In this case, layers are available for years 2018-2020, with year 2018 corresponding to the period November 2018 - October 2019. Unlike the microclimatic rasters of Šumava NP, non-forest areas in České Švýcarsko NP were cut out.

The following variables are available:

- 2018_FDD_baseT0_air_200cm_epsg32633.tif
- 2018_GDD_baseT5_air_200cm_epsg32633.tif
- 2018_T_max95p_air_15cm_epsg32633.tif
- 2018_T_max95p_air_200cm_epsg32633.tif
- 2018_T_mean_air_15cm_epsg32633.tif
- 2018_T_mean_air_200cm_epsg32633.tif
- 2018_T_mean_soil_8cm_epsg32633.tif
- 2018_T_min5p_air_200cm_epsg32633.tif
- 2018_snow_sum_epsg32633.tif

(for other years, only the beginning of the name changes)

```
# URL for file download (file T.air_200_cm.mean.tif from Zenodo)
soubor<-"2020_T_mean_air_15cm_epsg32633.tif"
url <- paste0("https://gitlab.ibot.cas.cz/",
              "matej.man/microclimate-atlas-public/-/",
              "raw/main/geodata/EPSG32633/",soubor)

# Path where to save file
destfile <- paste0("./data/rastry/", soubor)

# Create folder if it doesn't exist
dir.create("./data/rastry", recursive = TRUE, showWarnings = FALSE)

# Download file if it doesn't exist
if (!file.exists(destfile)) {
  download.file(url, destfile, mode = "wb")
}

# Loading raster using terra library
npcs <- terra::rast(destfile)
```

Map

Let's look at a map of one layer again. Once we have the map loaded as a raster, all subsequent steps are the same as in chapter 4.1.

```
# Setting space for legend
par(mar = c(5, 4, 4, 5))  # Adding space on right side for legend
par(oma = c(2, 1, 1, 1))  # Increasing outer bottom margin

# Basic plot of raster using plot() function
plot(npcs, col = viridis::viridis(100),
     xlab = "Longitude", ylab = "Latitude")

# Adding label to legend with unit (°C)
mtext("Temperature (°C)", side = 4, line = 3)  # Adding label on right side
```

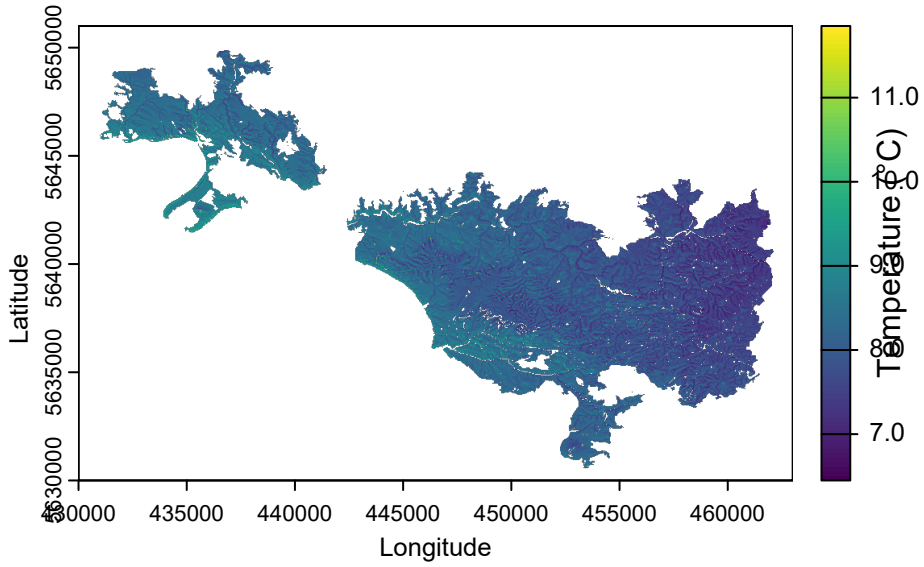

Figure 48: Simple map shows average temperature for the period November 2020 - October 2021.

References

BOX, G. E. P., and JENKINS, G. M. (1970) Time Series Analysis: Forecasting and Control. San Francisco: Holden-Day.

BRŮNA, J., KLINEROVÁ, T., KONOPOVÁ, Z., KALČÍK, V. a KIRSCHNER, J. (2023) Využití mikroklimatických dat v památkách zahradního umění. *Zahradnictví*. vol. 11.

BRŮNA, J., MACEK, M., MAN, M., HEDEROVÁ, L., KLINEROVÁ, T., MOUDRÝ, V., HEURICH, M., ČERVENKA, J., WILD, J. a KOPECKÝ, M. (2023) High-resolution microclimatic grids for the Bohemian Forest Ecosystem (1.0) [Data set]. *Zenodo*. doi: https://doi.org/10.5281/zenodo.6352641

BRŮNA J., WILD J., HEDEROVÁ L., KLINEROVÁ T., MACEK M. a URBANOVÁ R. (2021) Metodika měření mikroklimatu pomocí mikroklimatických stanic systému TMS. *Botanický ústav AVČR* http://hdl.handle.net/11104/0317943

CHIANUCCI F. a MACEK M. (2023) hemispheR: an R package for fisheye canopy image analysis. *Agricultural and Forest Meteorology*. vol. 336: 109470. doi: https://doi.org/10.1016/j.agrformet.2023.109470

CRAWLEY, M. J. (2013). The R Book. John Wiley & Sons.

CRUZ-ALONZO, V., PUCHER, C., RARCLIFFE, S., RUIZ-BENITO, P., ASTIGARRAGA, J., NEUMANN, M., HASENAUER, H., RODRÍGUEZ-SÁNCHEZ, F. (2023) "The easyclimate R package: Easy access to high-resolution daily climate data for Europe." *Environmental Modelling & Software*, 105627. doi: https://doi.org/10.1016/j.envsoft.2023.105627.

HAESEN, S., et al. (2021) ForestTemp – Sub-canopy microclimate temperatures of European forests. *Global Change Biology*. vol. 27(23), s. 6307–6319. doi: https://doi.org/10.1111/gcb.15892

HAESEN, S., et al. (2023) ForestClim – Bioclimatic variables for microclimate temperatures of European forests. *Global Change Biology*. vol. 29: s. 2886-2892 doi: https://doi.org/10.1111/gcb.16678

LEMBRECHTS J.J., et al. (2022) Global maps of soil temperature. *Global Change Biology* vol. 28: s. 3110–3144. doi: https://doi.org/10.1111/gcb.16060

KOPECKÝ, M., MACEK, M. a WILD, J. (2021) Topographic Wetness Index calculation guidelines based on measured soil moisture and plant species composition. *Science of the Total Environment*. vol. 757, 143785. doi: https://doi.org/10.1016/j.scitotenv.2020.143785

MACEK, M., KOPECKÝ, M., a WILD, J. (2019) Maximum air temperature controlled by landscape topography affects plant species composition in temperate forests. *Landscape Ecology*. vol. 34: s. 2541–2556. doi: https://doi.org/10.1007/s10980-019-00903-x

MAN, M., KALČÍK, V., MACEK, M., BRŮNA, J., HEDEROVÁ, L., WILD, J. a KOPECKÝ, M. (2023) myClim: Microclimate data handling and standardised analyses in R. *Methods in Ecology and Evolution*. vol. 14: s. 2308-2320. doi: https://doi.org/10.1111/2041-210X.14192

MORENO, A., HASENAUEER, H. (2016) Spatial downscaling of European climate data. *International Journal of Climatology*, 1444–1458. doi: https://doi.org/10.1002/joc.4436

NOAA National Centers of Environmental Information (1999) Global Surface Summary of the Day - GSOD. 1.0. *NOAA National Centers for Environmental Information.* id: gov.noaa.ncdc:C00516

PUCHER, C., NEUMANN, M. (2022). Description and Evaluation of Downscaled Daily Climate Data Version 3. *figshare* doi: https://doi.org/10.6084/m9.figshare.19070162.v1

PUCHER, C. (2023). Description and Evaluation of Downscaled Daily Climate Data Version 4. *figshare* doi: https://doi.org/10.6084/m9.figshare.22962671.v1

SPARKS, A. H., HENGL, T. a NELSON, A. (2017) GSODR: Global Summary Daily Weather Data in R. *The Journal of Open Source Software*, 2(10). doi: https://doi.org/10.21105/joss.00177

WILD, J., KIRSCHNER, J., MORAVEC, D. a KOHLOVÁ, J. (2014) Microclimate measurement as one of the prerequisites for succesful introduction of ornamental trees. *Acta Pruhoniciana*. vol. 108: s. 5–13.

WILD, J., KOPECKÝ, M., MACEK, M., ŠANDA, M., JANKOVEC, J. a T. HAASE. (2019) Climate at ecologically relevant scales: A new temperature and soil moisture logger for long-term microclimate measurement. *Agricultural and Forest Meteorology*. vol. 268: s. 40–47. doi: https://doi.org/10.1016/j.agrformet.2018.12.018