OPTAIN

Optimal Strategies to Retain Water and Nutrients

# D3.3: Created data pre-processors successfully applied for input data restructuring

Authors: Natalja Čerkasova (KU), Attila Nemes (NIBIO)

Co-Authors: Brigitta Szabó (ATK), Rasa Idzelytė (KU), Gökhan Cüceloğlu (KU), János Mészáros (ATK), Piroska Kassai (ATK), Moritz Shore (NIBIO), Csilla Farkas (NIBIO), Levente Czelnai (ATK)

Delivery Date: 31. August 2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant agreement No. 862756.

## Disclaimer

## Intellectual Property Rights

## Project Consortium

# Document Information

| | |
|---|---|
| Program | EU Horizon 2020 Research and Innovation Action H2020-EU.3.2.1.1 (SFS-23-2019) |
| Grant agreement No. | 862756 |
| Project acronym | OPTAIN |
| Project full name | Optimal strategies to retain and re-use water and nutrients in small agricultural catchments across different soil-climatic regions in Europe |
| Start of the project | September 2020 |
| Duration | 60 months |
| Project coordination | Prof. Dr. Martin Volk<br>Helmholtz-Centre for Environmental Research GmbH - UFZ<br>www.optain.eu |
| Deliverable | D3.3: Created data pre-processors successfully applied for input data restructuring<br>Description of the pre-processing scripts and routines for the data harmonisation for the usage as inputs fitting the needs of modelling approaches. |
| Work package | WP3: Retrieval of modelling data and solutions to overcome data scarcity |
| Task | Task 3.3: Overcoming data scarcity |
| Lead beneficiary | Klaipeda University |
| Author(s) | Natalja Čerkasova (KU, partner no. 11), Attila Nemes (NIBIO, partner no. 14) |
| Contributor(s) | Brigitta Szabó (ATK), Rasa Idzelytė (KU), Gökhan Cüceloğlu (KU), János Mészáros (ATK), Piroska Kassai (ATK), Moritz Shore (NIBIO), Csilla Farkas (NIBIO), Levente Czelnai (ATK) |
| Quality check | Felix Witing (UFZ), Martin Volk (UFZ) |
| Planned delivery date | Month 24 (August 2022) |
| Actual delivery date | 31/08/2022 |
| Citation | Čerkasova, N., Nemes, A., Szabó, B., Idzelytė, R., Cüceloğlu, G., Mészáros, J., Kassai, P., Shore, M., Farkas, C. & L. Czelnai (2022): *Created data pre-processors successfully applied for input data restructuring*. Deliverable D3.3 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756 |
| Dissemination level* | PU |

*PU = Public; PP = Restricted to other program participants (including the Commission Services; CO = Confidential, only for members of the consortium (including the Commission Services).

## Deliverable status

| Version | Date | Author(s)/Contributor(s) | Notes |
|---|---|---|---|
| 0.7 | 01.08.2022 | Natalja Čerkasova (KU), Attila Nemes (NIBIO) | First complete draft |
| 0.8 | 16.08.2022 | Martin Volk (UFZ), Felix Witing (UFZ) | First Revision |
| 0.9 | 23.08.2022 | Natalja Čerkasova (KU) | Second draft |
| 1.0 | 26.08.2022 | Martin Volk (UFZ), Natalja Čerkasova (KU), Felix Witing (UFZ) | Quality check and Final revision |

## Summary

The OPTAIN project aims to identify efficient measures for the retention and reuse of water and nutrients (NSWRM - Natural/Small Water Retention Measures) in small agricultural catchments based on empirical data and scale-adapted integrated modelling approaches. The project involves international partners with case study sites in 14 small agricultural catchments (including one cross-border), all having different data availability, measurement protocols, data handling policies and formats. Based on the agreed data harmonisation procedures within the OPTAIN project, this deliverable D3.3 provides data pre-processors for input data restructuring to overcome the aforementioned differences among the partners. The projects' case study leaders collected the input data necessary for the modelling tasks structured according to the derived guidelines. Available input information from different sources (both national and global or European scale) and formats had to be harmonised and standardised where relevant and reasonable. Pre-processing tools have been developed, which were used for data compilation and reformatting of the input data in line with the needs of basin-scale (SWAT+) and the field scale (SWAP) modelling approaches. Freely available and distributable software, programming languages, and technologies (Python, R, JavaScript) were used for these tasks.

# Table of Contents

Summary ........ 5
Abbreviations ........ 7
1. Introduction ........ 8
2. Model input data pre-processing ........ 9
2.1. Field-scale model (SWAP) ........ 9
2.1.1. Daily meteorological data format ........ 9
2.1.2. Reference data quality check ........ 11
2.1.3. Soft-calibration and visualisation tool ........ 14
2.1.4. Sub-daily meteorological data format ........ 17
2.1.5. Autocalibration tool ........ 18
2.2. Catchment-scale model (SWAT) ........ 20
2.2.1. Future meteorological data ........ 20
2.2.2. Soil physical and hydrological properties ........ 21
2.2.3. Soil phosphorus content ........ 22
2.2.4. Crop map pre-processing ........ 23
2.3. Metadata for model inputs ........ 24
2.3.1. Spreadsheet-based approach ........ 25
2.3.2. Script-based approach ........ 25
References ........ 27
Appendixes ........ 29
1. Script for creating meteorological input files for SWAP in daily resolution ........ 29
2. Script for reference data quality check ........ 31
3. SWAP soft calibration and visualisation tool ........ 43
4. Script for creating meteorological input files for SWAP in sub-daily resolution 54
5. Batch File to run SWAP model and postprocessor ........ 57
6. Script to Read SWAP Output ........ 58
7. Computation of soil physical and hydrological properties ........ 62
8. Script to map soil phosphorus content ........ 66
9. Script to add soil phosphorus content to Hydrologic Response Units ........ 75
10. Google Earth Engine script to build and apply crop classification model ........ 77
11. Script to merge land use and crop map ........ 86
12. Script to derive metadata XML for multiple data files ........ 88

# Abbreviations

| | |
|---|---|
| CS | Case Study |
| CSL | Case Study Lead |
| CSS | Case Study Site |
| GIS | Geographic Information System |
| GUI | Graphical User Interface |
| NetCDF | Network Common Data Form. Set of software libraries and self-describing, machine-independent data formats |
| NSE | Nash-Sutcliffe Efficiency |
| NSWRM | Natural/Small Water Retention Measures |
| PBIAS | Percent bias |
| PEST | Model-Independent Parameter Estimation and Uncertainty Analysis Tool |
| Python | High-level, interpreted, general-purpose programming language |
| R | Programming language for statistical computing and graphics |
| RSR | Root-Squared Error |
| WP | Work Package |
| SWAP | Soil Water Atmosphere Plant model |
| SWAT | Soil and Water Assessment Tool |
| SWC | Soil Water Content |
| XML | Extensible Markup Language |

# 1. Introduction

The OPTAIN project aims to identify efficient measures for the retention and reuse of water and nutrients (NSWRM - Natural/Small Water Retention Measures) in small agricultural catchments across continental, pannonian and boreal regions of Europe, based on empirical data and scale-adapted integrated modelling. The project focuses on 14 case study sites (CSS) in various countries (one case study (CS) is transboundary) and hydroclimatic conditions, all having different data availability, measurement protocols, data handling policies, formats and standards. To enable a solid comparison of study site specific modelling results, we developed a harmonised approach for data sharing, processing, and result interpretation. The procedure is described in OPTAINs 'Knowledge Management Plan' (Witing & Volk, 2021) as well as in the 'Common Working Environment' (Čerkasova et al., 2021) with standardised metadata for the harmonized reporting of project outputs. Based on the agreed data harmonisation procedures within the OPTAIN project, this deliverable (D3.3) describes our developed tools and procedures (scripts) to help guide the case study leads (CSL) in restructuring the input data for modelling and reporting purposes, as well as developed solutions to overcome data scarcity (Szabó et al., 2022). Within the OPTAIN project, we use scripts to semi-automate many repetitive procedures for work optimisation, data harmonisation and/or supplementation.

As defined within the computer programming dictionary, a **script** is a computer language with several commands within a file (or several files) capable of being executed without being compiled (interpreted or carried out by another program rather than by the computer processor). In the context of this report, we consider a script as a sequence of commands, executed by another program/tool/GUI. To execute a script, a person has to have a certain degree of knowledge on scripting tools and languages, however basic understanding is sufficient. Moreover, as the tools are designed to deal with certain types and structures of data, the initial dataset has to be made ready for usage. Where appropriate, we provide guidance on how the initial data should look like and what are the scripts' capabilities.

This report presents the data processing tools (scripts) that we use during our work with data. The scripts are in use for tasks related to WP3 (Retrieval of modelling data and solutions to overcome data scarcity) and WP4 (Integrated assessment of NSWRM). We foresee that some of those scripts can be useful for the broader scientific community, hence we intend to publish or already published them in open access with supporting documentation. Every script, described in section 2 of this report, comes with a summary table for the benefit of the reader for fast interpretation of usage. This report is focused specifically on the scripts themselves, so we do not provide any scientific material or justification on the various methods that were used in the code.

# 2. Model input data pre-processing

Integrated and participatory modelling approaches play an important role in the OPTAIN project. They are used to combine and allocate Natural/Small Water Retention Measures (NSWRM) in a way that they respond best to the characteristics and management of a certain agricultural catchment under consideration of local constraints, other human uses and environmental needs. Within the OPTAIN project we develop both catchment-scale and field-scale models. The case studies for field-scale and catchment-scale models are listed in Table 1.

*Table 1. The case studies for application of field- and catchment-scale models.*

| No | Basin | Country | Field-scale | Catchment-scale |
|---|---|---|---|---|
| CS1 | Schwarzer Schöps | Germany | | + |
| CS2 | Petite Glâne | Switzerland | + | + |
| CS3a | Csorsza | Hungary | + | + |
| CS3b | Felső-Válicka | Hungary | | + |
| CS4 | Upper Zglowiaczka | Poland | + | + |
| CS5 | Pesnica | Austria/Slovenia | | + |
| CS6 | Kobiljski potok-Kebele | Slovenia/Hungary | | + |
| CS7 | La Wimbe | Belgium | | + |
| CS8 | Dotnuvėlė | Lithuania | + | + |
| CS9 | Cherio | Italy | | + |
| CS10 | Hobøl | Norway | + | + |
| CS11 | Tetves | Hungary | + | + |
| CS12 | Čechtický | Czech Republic | + | + |
| CS13 | Dviete | Latvia | | + |
| CS14 | Sävjaån | Sweden | | + |

The models require many different datasets (topography, weather observation data, crop yield and calendar, soil, land use, etc.) and formats (GIS shapefiles, raster files, text or database files, etc.) as input. Hence, we divide the following section based on the requirements of field and catchment-scale models.

## 2.1. Field-scale model (SWAP)

In OPTAIN, the effectiveness of NSWRMs at the field scale will be modeled by the process-based field-scale model SWAP. Currently (of August 2022), 7 case study sites are developing a field-scale model. The calibration and scenario analyses results will be cross-validated with those of the catchment-scale models.

### 2.1.1. Daily meteorological data format

Meteorological data are important input data for both model types applied in the OPTAIN project. To run the SWAP model the meteorological data needs to be in a specific format. The required format for daily weather data is shown in Figure 1. SWAP has various modes and settings, which determine which of the variables are required, and which are optional (see section 3.7 in Kroes et al., 2017). The data is stored in one text file per year, with a file extension equivalent to the last three digits of the respective year (i.e., “Meteofile.003” for year 2003). As this specific format is time-consuming and error-prone to be created manually, a script was created to automate the process by using a user-friendly Excel template (Figure 1). An automation of this process enables quick transformation of case-study specific weather data into SWAP readable format. Additionally, this allows for the possibility of running SWAP under multi-model ensemble modelling of the future climate for each case study, as this process would potentially require thousands of SWAP meteorological input files, which would otherwise be too time intensive to be created manually.

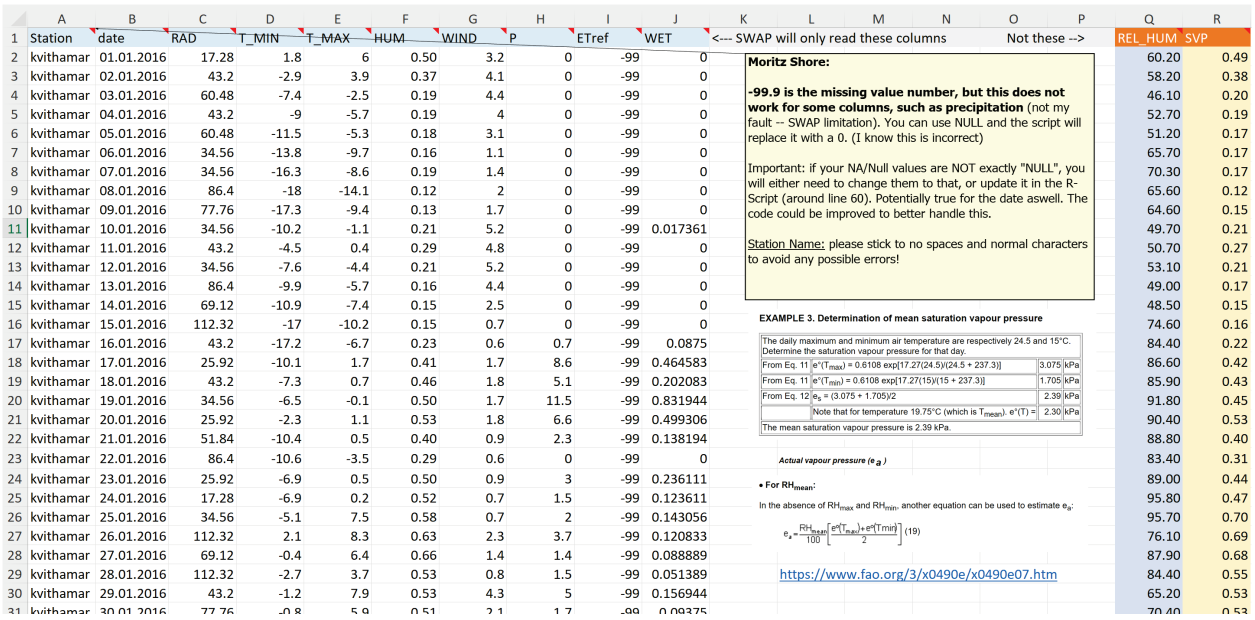
```
******************************************************************************************************
* Filename: Hupsel.003
* Contents: SWAP 4.0 - Daily meterorological data
*******************************************************************************************************
Comment area:
*                Station 283 = Hupsel
*
******************************************************************************************************
Station     DD     MM   YYYY     RAD     Tmin     Tmax       HUM     WIND     RAIN    ETref        WET
*           nr     nr     nr   kJ/m2       °C       °C       kPa      m/s       mm       mm          d
******************************************************************************************************
'283'        1      1   2003   650.0     -2.8     10.7      0.86      4.4      9.7      0.1     0.3625
'283'        2      1   2003   350.0      6.6     10.5      1.12      4.7     21.1      0.0     0.5250
'283'        3      1   2003   960.0      0.4      7.9      0.82      3.4      2.2      0.1     0.1125
'283'        4      1   2003  2290.0     -4.1      0.5      0.46      5.1      0.0      0.2     0.0000
```

*Figure 1. Text format of daily weather records required for the SWAP model. All lines starting with a “*” character are optional comment lines, not read by SWAP (Figure source: Kroes, J. G., et al, 2017). Where DD, MM and YYYY stand for day, month, and year; RAD – daily solar radiation; Tmin – daily minimum air temperature; Tmax – daily maximum air temperature; HUM – vapour pressure; WIND – wind speed; RAIN – daily rainfall amount; ETref – potential evapotranspiration; WET – ratio of the day with rainfall.*

To use the script, the user must first fill in the template Excel sheet with their data. The date range is entirely flexible. However, it must be limited to 1000 years in length due to the SWAP file extension system not being able to differentiate any file exactly 1000 years apart.

SWAP uses actual vapour pressure ($e_a$) instead of the more commonly used Relative Humidity (RH). The corresponding value of $e_a$ can be calculated from RH if daily minimum and maximum air temperatures (Tmin, Tmax) are provided. This can be done from within the template sheet, where more detailed instructions are provided (Figure 2, columns Q, R).

In SWAP, missing values are denoted by a value of “–99.9”. However, SWAP prevents any negative values for most meteorological variables (such as wind speed and precipitation), making this missing value identifier all but useless. No workaround has been found yet. Therefore, all data gaps in the Excel sheet should be filled in using, e.g. interpolation, before running the script.



|  | A | B | C | D | E | F | G | H | I | J | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Station | date | RAD | T_MIN | T_MAX | HUM | WIND | P | ETref | WET | REL_HUM | SVP |
| 2 | kvithamar | 01.01.2016 | 17.28 | 1.8 | 6 | 0.50 | 3.2 | 0 | -99 | 0 | 60.20 | 0.49 |
| 3 | kvithamar | 02.01.2016 | 43.2 | -2.9 | 3.9 | 0.37 | 4.1 | 0 | -99 | 0 | 58.20 | 0.38 |
| 4 | kvithamar | 03.01.2016 | 60.48 | -7.4 | -2.5 | 0.19 | 4.4 | 0 | -99 | 0 | 46.10 | 0.20 |
| 5 | kvithamar | 04.01.2016 | 43.2 | -9 | -5.7 | 0.19 | 4 | 0 | -99 | 0 | 52.70 | 0.19 |
| 6 | kvithamar | 05.01.2016 | 60.48 | -11.5 | -5.3 | 0.18 | 3.1 | 0 | -99 | 0 | 51.20 | 0.17 |
| 7 | kvithamar | 06.01.2016 | 34.56 | -13.8 | -9.7 | 0.16 | 1.1 | 0 | -99 | 0 | 65.70 | 0.17 |
| 8 | kvithamar | 07.01.2016 | 34.56 | -16.3 | -8.6 | 0.19 | 1.4 | 0 | -99 | 0 | 70.30 | 0.17 |
| 9 | kvithamar | 08.01.2016 | 86.4 | -18 | -14.1 | 0.12 | 2 | 0 | -99 | 0 | 65.60 | 0.12 |
| 10 | kvithamar | 09.01.2016 | 77.76 | -17.3 | -9.4 | 0.13 | 1.7 | 0 | -99 | 0 | 64.60 | 0.15 |
| 11 | kvithamar | 10.01.2016 | 34.56 | -10.2 | -1.1 | 0.21 | 5.2 | 0 | -99 | 0.017361 | 49.70 | 0.21 |
| 12 | kvithamar | 11.01.2016 | 43.2 | -4.5 | 0.4 | 0.29 | 4.8 | 0 | -99 | 0 | 50.70 | 0.27 |
| 13 | kvithamar | 12.01.2016 | 34.56 | -7.6 | -4.4 | 0.21 | 5.2 | 0 | -99 | 0 | 53.10 | 0.21 |
| 14 | kvithamar | 13.01.2016 | 86.4 | -9.9 | -5.7 | 0.16 | 4.4 | 0 | -99 | 0 | 49.00 | 0.17 |
| 15 | kvithamar | 14.01.2016 | 69.12 | -10.9 | -7.4 | 0.15 | 2.5 | 0 | -99 | 0 | 48.50 | 0.15 |
| 16 | kvithamar | 15.01.2016 | 112.32 | -17 | -10.2 | 0.15 | 0.7 | 0 | -99 | 0 | 74.60 | 0.16 |
| 17 | kvithamar | 16.01.2016 | 43.2 | -17.2 | -6.7 | 0.23 | 0.6 | 0.7 | -99 | 0.0875 | 84.40 | 0.22 |
| 18 | kvithamar | 17.01.2016 | 25.92 | -10.1 | 1.7 | 0.41 | 1.7 | 8.6 | -99 | 0.464583 | 86.60 | 0.42 |
| 19 | kvithamar | 18.01.2016 | 43.2 | -7.3 | 0.7 | 0.46 | 1.8 | 5.1 | -99 | 0.202083 | 85.90 | 0.43 |
| 20 | kvithamar | 19.01.2016 | 34.56 | -6.5 | -0.1 | 0.50 | 1.7 | 11.5 | -99 | 0.831944 | 91.80 | 0.45 |
| 21 | kvithamar | 20.01.2016 | 25.92 | -2.3 | 1.1 | 0.53 | 1.8 | 6.6 | -99 | 0.499306 | 90.40 | 0.53 |
| 22 | kvithamar | 21.01.2016 | 51.84 | -10.4 | 0.5 | 0.40 | 0.9 | 2.3 | -99 | 0.138194 | 88.80 | 0.40 |
| 23 | kvithamar | 22.01.2016 | 86.4 | -10.6 | -3.5 | 0.29 | 0.6 | 0 | -99 | 0 | 83.40 | 0.31 |
| 24 | kvithamar | 23.01.2016 | 25.92 | -6.9 | 0.5 | 0.50 | 0.9 | 3 | -99 | 0.236111 | 89.00 | 0.44 |
| 25 | kvithamar | 24.01.2016 | 17.28 | -6.9 | 0.2 | 0.52 | 0.7 | 1.5 | -99 | 0.123611 | 95.80 | 0.47 |
| 26 | kvithamar | 25.01.2016 | 34.56 | -5.1 | 7.5 | 0.58 | 0.7 | 2 | -99 | 0.143056 | 95.70 | 0.70 |
| 27 | kvithamar | 26.01.2016 | 112.32 | 2.1 | 8.3 | 0.63 | 2.3 | 3.7 | -99 | 0.120833 | 76.10 | 0.69 |
| 28 | kvithamar | 27.01.2016 | 69.12 | -0.4 | 6.4 | 0.66 | 1.4 | 1.4 | -99 | 0.088889 | 87.90 | 0.68 |
| 29 | kvithamar | 28.01.2016 | 112.32 | -2.7 | 3.7 | 0.53 | 0.8 | 1.5 | -99 | 0.051389 | 84.40 | 0.55 |
| 30 | kvithamar | 29.01.2016 | 43.2 | -1.2 | 7.9 | 0.53 | 4.3 | 5 | -99 | 0.156944 | 65.20 | 0.53 |
| 31 | kvithamar | 30.01.2016 | 77.76 | -0.8 | 5.9 | 0.51 | 2.1 | 1.7 | -99 | 0.09375 | 70.40 | 0.53 |

<--- SWAP will only read these columns

Not these -->

**Moritz Shore:**

**-99.9 is the missing value number, but this does not work for some columns, such as precipitation** (not my fault -- SWAP limitation). You can use NULL and the script will replace it with a 0. (I know this is incorrect)

Important: if your NA/Null values are NOT exactly "NULL", you will either need to change them to that, or update it in the R-Script (around line 60). Potentially true for the date aswell. The code could be improved to better handle this.

Station Name: please stick to no spaces and normal characters to avoid any possible errors!

**EXAMPLE 3. Determination of mean saturation vapour pressure**

| | | | |
|---|---|---|---|
| The daily maximum and minimum air temperature are respectively 24.5 and 15°C. Determine the saturation vapour pressure for that day. | | | |
| From Eq. 11 | $e°(T_{max}) = 0.6108 \exp[17.27(24.5)/(24.5 + 237.3)]$ | 3.075 | kPa |
| From Eq. 11 | $e°(T_{min}) = 0.6108 \exp[17.27(15)/(15 + 237.3)]$ | 1.705 | kPa |
| From Eq. 12 | $e_s = (3.075 + 1.705)/2$ | 2.39 | kPa |
| | Note that for temperature 19.75°C (which is $T_{mean}$). e°(T) = | 2.30 | kPa |
| The mean saturation vapour pressure is 2.39 kPa. | | | |

***Actual vapour pressure ($e_a$)***

• **For $RH_{mean}$:**

In the absence of $RH_{max}$ and $RH_{min}$, another equation can be used to estimate $e_a$:

$$e_a = \frac{RH_{mean}}{100}\left[\frac{e°(T_{max}) + e°(T min)}{2}\right] \quad (19)$$

https://www.fao.org/3/x0490e/x0490e07.htm

*Figure 2. Excel template sheet for meteorological data. Data supplied here is read by the script and converted into SWAP compatible format. Each column contains a note describing its purpose.*

Once the Excel template sheet has been filled in with desired data, the script can be run. The script prompts the user to select the location of the template sheet using a GUI file explorer, followed by the desired location of the output files. After these two steps have been taken, the script generates the SWAP compatible output files to the desired location. The R-script is documented, and the Excel template sheet provides notes to describe its function. Additionally, a more detailed readme file is available providing step-by-step instructions. The script, template, readme, and example output are stored in the OPTAIN cloud (accessible for all OPTAIN partners) under the following path: *[/WPs & Tasks/WP3/Scripts and tools/Met Data Daily Resolution]*

*Table 2. Summary of the script for creating formatted daily meteorological data for field-scale models.*

| | |
|---|---|
| Source data structure: | Excel file, *.xlsx |
| Processed data structure: | Text files for SWAP input |
| In use for CS: | Switzerland (CS 2), Hungary (CS 3, 11), Poland (CS 4), Norway (CS 10), Czech Rep. (CS 12) |
| Language or tool: | R (v 2021.09.1) |
| Script availability: | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 1 |

### 2.1.2. Reference data quality check

The calibration of the SWAP model requires reference data that needs to be compared with the model outputs. In case of soil hydrological models, the most important reference data is the soil water content, which could be complemented with surface runoff, drainage outflow and soil temperature.

Soil water content (SWC) is traditionally measured by different sensors/probes installed in the representative soil layers. The measurement accuracy of such probes depends on the sensor technique, which is sensitive to soil characteristics such as soil texture, temperature, bulk density, and salinity. However, the calibration functions provided by instrument manufacturers are generally developed under laboratory conditions, and their accuracy for field applications is rarely investigated (Mittelbach et al., 2012). Studies comparing the performance of such sensors concluded that i) sensors from different manufacturers can accurately depict soil moisture dynamics of the reference method, but can differ significantly in absolute records, and ii) in-field calibration of soil moisture probes is highly recommended (Mittelbach et al., 2012; Leib et al., 2003). This indicates that a reference data quality check is needed before starting the model calibration. Besides, soil moisture content data measured in frozen soil needs to be removed, as the sensors measure the liquid part of the water in the soil, whilst the model calculates the total amounts. Thus, the measured and simulated SWC values are not comparable for frozen soils.

During the reference data quality check, we check if:

- Soil temperature dynamics is in line with the air temperature (graphical check)
- SWC should not be far below the wilting point (water content corresponding to water potential of pF=4.2, Figure 3)
- SWC should not be much higher than the total porosity / saturated water content of the soil (Figure 3)
- SWC for periods when the soil is freezing should be removed. The low values can be removed via a cross-check with the soil temperature of the same layer, but a graphical representation of soil water content dynamics clearly shows the values that should be blacklisted.

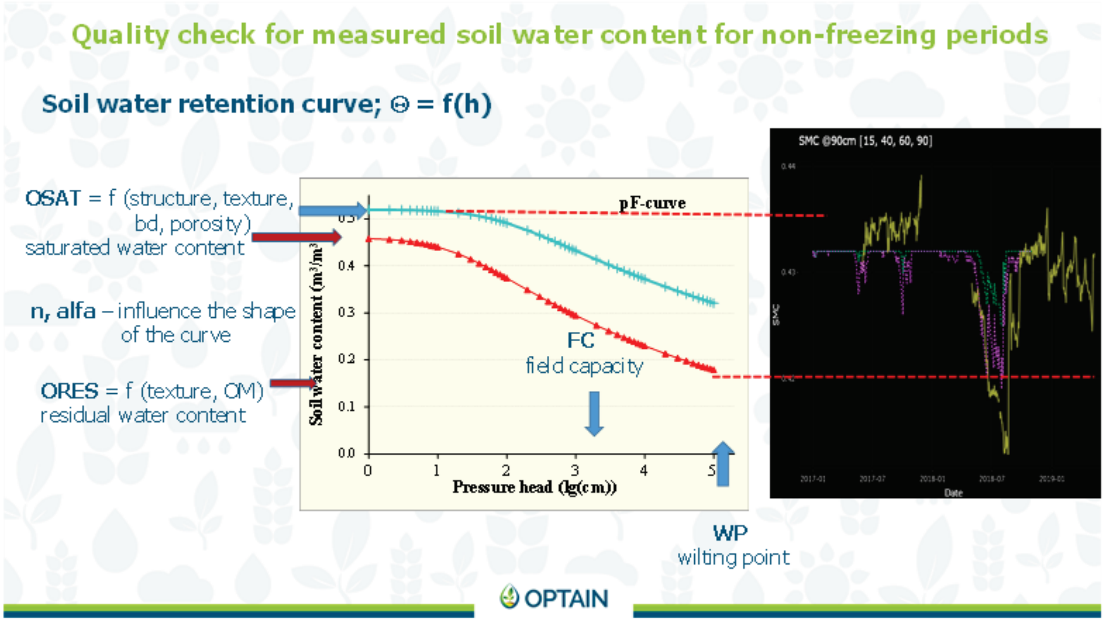
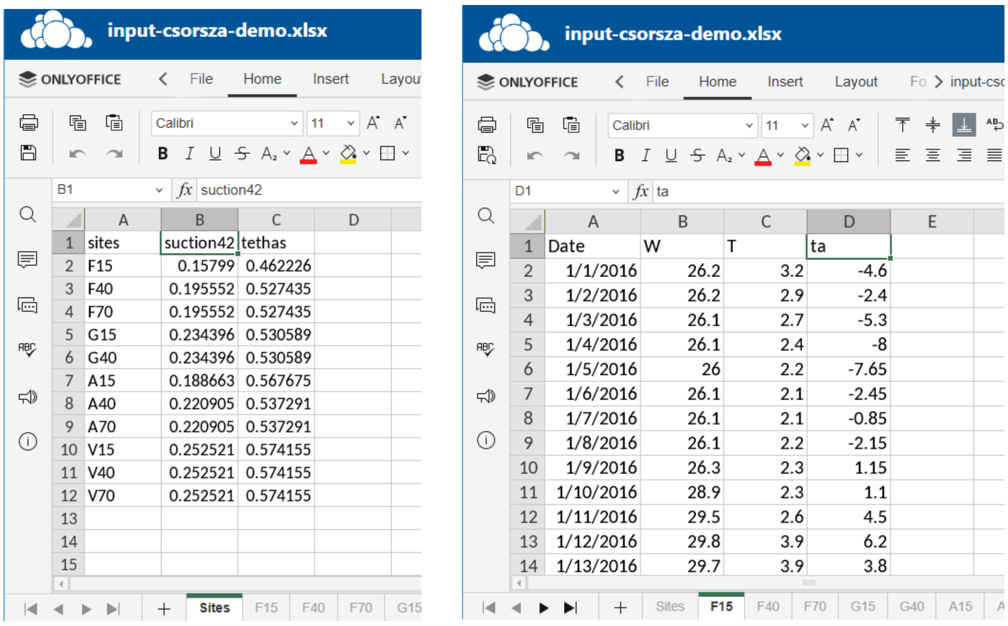


Figure 3. Graphical presentation of the quality check for observed soil water content



input-csorsza-demo.xlsx (left, sheet Sites, B1: suction42)

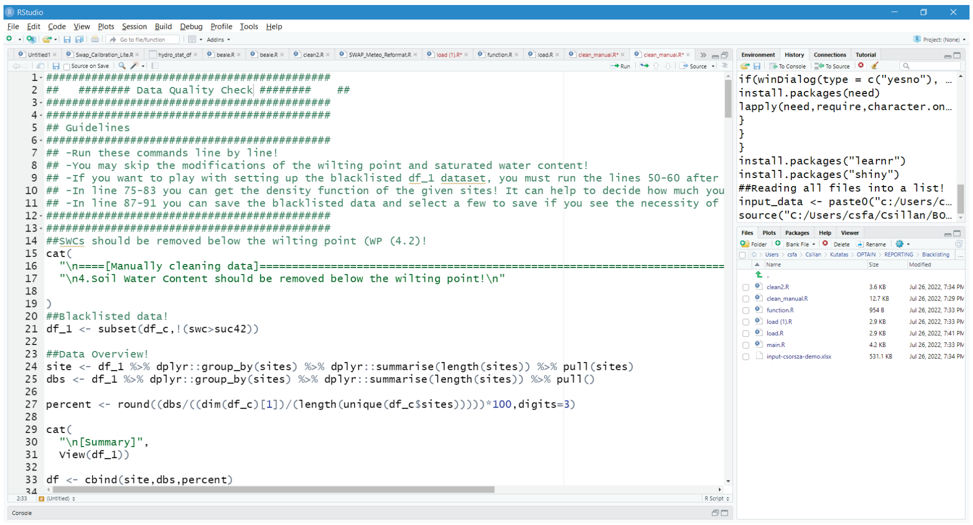
| | A | B | C |
|---|---|---|---|
| 1 | sites | suction42 | tethas |
| 2 | F15 | 0.15799 | 0.462226 |
| 3 | F40 | 0.195552 | 0.527435 |
| 4 | F70 | 0.195552 | 0.527435 |
| 5 | G15 | 0.234396 | 0.530589 |
| 6 | G40 | 0.234396 | 0.530589 |
| 7 | A15 | 0.188663 | 0.567675 |
| 8 | A40 | 0.220905 | 0.537291 |
| 9 | A70 | 0.220905 | 0.537291 |
| 10 | V15 | 0.252521 | 0.574155 |
| 11 | V40 | 0.252521 | 0.574155 |
| 12 | V70 | 0.252521 | 0.574155 |

input-csorsza-demo.xlsx (right, sheet F15, D1: ta)

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Date | W | T | ta |
| 2 | 1/1/2016 | 26.2 | 3.2 | -4.6 |
| 3 | 1/2/2016 | 26.2 | 2.9 | -2.4 |
| 4 | 1/3/2016 | 26.1 | 2.7 | -5.3 |
| 5 | 1/4/2016 | 26.1 | 2.4 | -8 |
| 6 | 1/5/2016 | 26 | 2.2 | -7.65 |
| 7 | 1/6/2016 | 26.1 | 2.1 | -2.45 |
| 8 | 1/7/2016 | 26.1 | 2.1 | -0.85 |
| 9 | 1/8/2016 | 26.1 | 2.2 | -2.15 |
| 10 | 1/9/2016 | 26.3 | 2.3 | 1.15 |
| 11 | 1/10/2016 | 28.9 | 2.3 | 1.1 |
| 12 | 1/11/2016 | 29.5 | 2.6 | 4.5 |
| 13 | 1/12/2016 | 29.8 | 3.9 | 6.2 |
| 14 | 1/13/2016 | 29.7 | 3.9 | 3.8 |

*Figure 4. Wilting point (suction42) and saturated water content (tethas), corresponding to different soil layers of the reference sites. F, G, A, V indicate the site name, 15, 40 etc. the depth of the soil moisture probes (left); Reference worksheet for observed data. W- soil water content, T- soil temperature, ta – air temperature (right)*

The reference data quality check was automatized in an R environment. The input file for running the code is an Excel workbook, containing i) the ID of the soil layer and corresponding wilting point and saturated water content (Worksheet 1, Figure 4, left); ii) the observed data to be checked (Worksheets 2-11, Figure 4, right).



```
#############################################
##   ######## Data Quality Check ########    ##
#############################################
#############################################
## Guidelines
#############################################
## -Run these commands line by line!
## -You may skip the modifications of the wilting point and saturated water content!
## -If you want to play with setting up the blacklisted df_1 dataset, you must run the lines 50-60 after
## -In line 75-83 you can get the density function of the given sites! It can help to decide how much you
## -In line 87-91 you can save the blacklisted data and select a few to save if you see the necessity of
#############################################
#############################################
##SWCs should be removed below the wilting point (WP (4.2)!
cat(
  "\n====[Manually cleaning data]==================================================
  "\n4.Soil Water Content should be removed below the wilting point!\n"

)
##Blacklisted data!
df_1 <- subset(df_c,!(swc>suc42))

##Data Overview!
site <- df_1 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull(sites)
dbs <- df_1 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull()

percent <- round((dbs/((dim(df_c)[1])/(length(unique(df_c$sites)))))*100,digits=3)

cat(
  "\n[Summary]",
  View(df_1))

df <- cbind(site,dbs,percent)
```

```
if(winDialog(type = c("yesno"), …
install.packages(need)
lapply(need,require,character.on…
}
}
}
install.packages("learnr")
install.packages("shiny")
##Reading all files into a list!
input_data <- paste0("c:/Users/c…
source("C:/Users/csfa/Csillan/BO…
```

C: > Users > csfa > Csillan > Kutatas > OPTAIN > REPORTING > Blacklisting

| Name | Size | Modified |
|---|---|---|
| clean2.R | 3.6 KB | Jul 26, 2022, 7:34 PM |
| clean_manual.R | 12.7 KB | Jul 26, 2022, 7:29 PM |
| function.R | 954 B | Jul 26, 2022, 7:33 PM |
| load (1).R | 2.9 KB | Jul 26, 2022, 7:33 PM |
| load.R | 2.9 KB | Jul 26, 2022, 7:41 PM |
| main.R | 4.2 KB | Jul 26, 2022, 7:33 PM |
| input-csorsza-demo.xlsx | 531.1 KB | Jul 26, 2022, 7:34 PM |

*Figure 5. Inbuilt instructions for running the R-code for data quality check*

The R-script incorporates direct instructions on how to adjust and run the code (Figure 5). The data series can be plotted for visual analyses. The demo version, which is being upgraded up to OPTAIN partner's requests, is available on the OPTAIN cloud under the following path:
*/WPs & Tasks/WP3/Scripts and tools/App2_Reference_data_Quality_Check.r*

*Table 3. Summary of the script for checking the reference data quality for field-scale models.*

| | |
|---|---|
| Source data structure: | Excel workbook *.xlsx |
| Processed data structure: | Excel files *.xlsx, containing the corrected reference data and used as input for the SWAP soft-calibration tool |
| In use for CS: | Switzerland (CS 2), Hungary (CS 3, 11), Poland (CS 4), Lithuania (CS 8), Norway (CS 10), Czech Rep. (CS 12) |
| Language or tool: | R (v 2021.09.1) |
| Script availability: | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 2 |

### 2.1.3. Soft-calibration and visualisation tool

The SWAP model does not feature a graphical user interface (GUI).  It runs by command line and all inputs and outputs are stored as text files. These characteristics make it partly difficult to apply the SWAP model, but at the same time easy to connect the model to other systems. The unprocessed output of SWAP is hard to read, even more when comparing it to measured reference data, or previous model runs under differing parameter settings. Additionally, the evaluation of the model performance requires statistical indicators, such as the Nash-Sutcliffe Efficiency (NSE) (Nash & Sutcliffe 1970), which must be calculated by comparing measured reference data to modelled data. As SWAP does not have the inherit ability to accomplish these vital aspects of model calibration and validation, work was begun on an R-based environment for soft calibration and visualization.

The R-based environment executes the SWAP model, reads the output and reformats it into a standardized format. To compare model output with measured reference data (such as soil moisture), the depth of the measured reading must match the depth of the modelled output value. If this is not the case, the script recalculates the modelled values by picking the closest soil depth or averaging two soil depths if they are equidistant (the utilization of a more advanced method of interpolation is under consideration). To ensure compatibility with all the case study sites, this function was built in a flexible manner and can therefore handle an unlimited quantity of measurements at any desired soil depth. An automatic backup of the processed output, as well as of the input files and model parameters are created and stored in a dedicated directory. This allows for intercomparison of model runs as well as the ability to undo to a previous model setup.

With the SWAP output reformatted and harmonized with the site-specific measurements, statistical indicators of model performance can be calculated. Currently we use the aforementioned NSE, as well as $R^2$, PBIAS, and RSR (Moriasi et al., 2015). Additionally, and most importantly for manual soft calibration, the model output is plotted against the measured values to provide visual feedback on model performance, as seen in Figure 6. The user can choose to compare any number of past archived model runs with the current model set-up, thus being able to identify changes resulting from alterations made to the model setup. This can also be seen in Figure 6. Usage of the R package "plotly" allows for interaction with the plot, including zooming, panning, comparison of point values, hiding specific time series, and more.

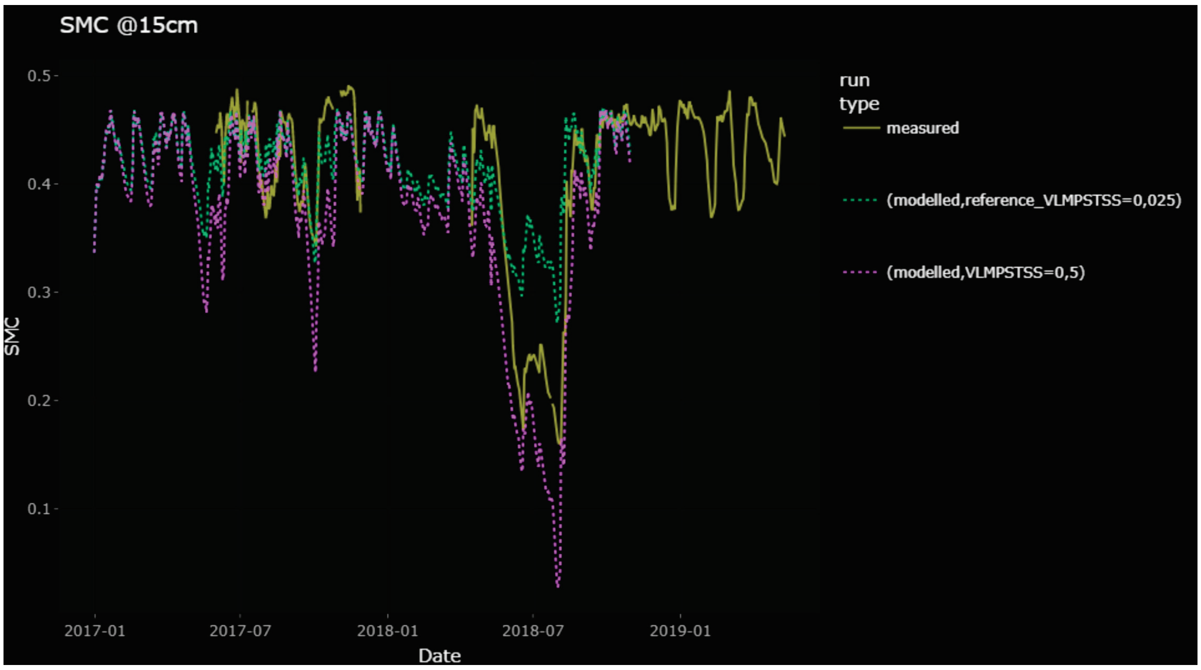


*Figure 6. Visualized output of soil moisture content (SMC) of the measured reference data (solid line) and SWAP model runs (dotted lines) at 15 cm soil depth. The green line is the latest model run, the purple line the previous model run. The model run name was altered to convey which parameter was changed. In this case an alteration of parameter VLMPSTSS from 0.5 to 0.025 reduced model skill in predicting the drop in SMC during July 2018.*

The calculated statistical indicators can be compared to previous model runs (Figure 7). This can be done for all indicators and all soil depths, as well as for an average of all depths. Comparison of statistical indicators allows the user to see which model setup had the best performance, and by how much it was improved.

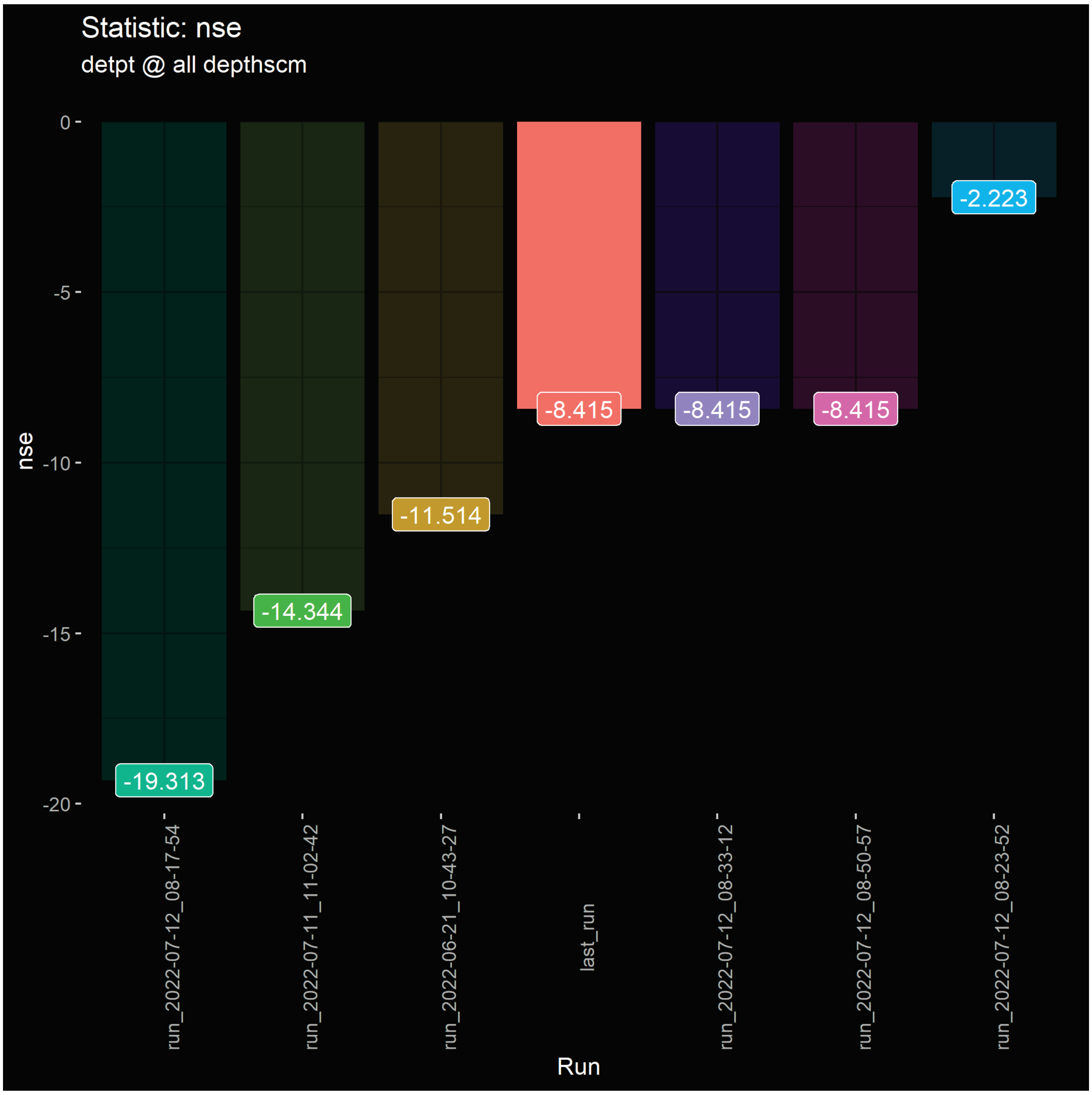


*Figure 7. Graphical comparison of NSE values of various runs. The solid-coloured column indicates the current model setup. Default nomenclature for previous model runs is the creation date, however this can be renamed as desired.*

Finally, the script opens the SWAP main file, allowing the user to make any desired alterations. This file can be saved and closed, and upon running the script once more, the altered setup will be added to the analysis, repeating the cycle. The model setups to be included in the analysis is controlled by adding or removing archives of the model setups in the archive directory.

This tool in its current state is functional and useable, yet still predominantly a proof of concept, and not in a finished state. More features are planned, such as comparison to other measurements and soil properties (i.e. precipitation events, pressure head). The tool is currently slow and inefficient, but this will be fixed in future releases. Moreover, ease of use will be improved, and a documentation will be provided in the finalized

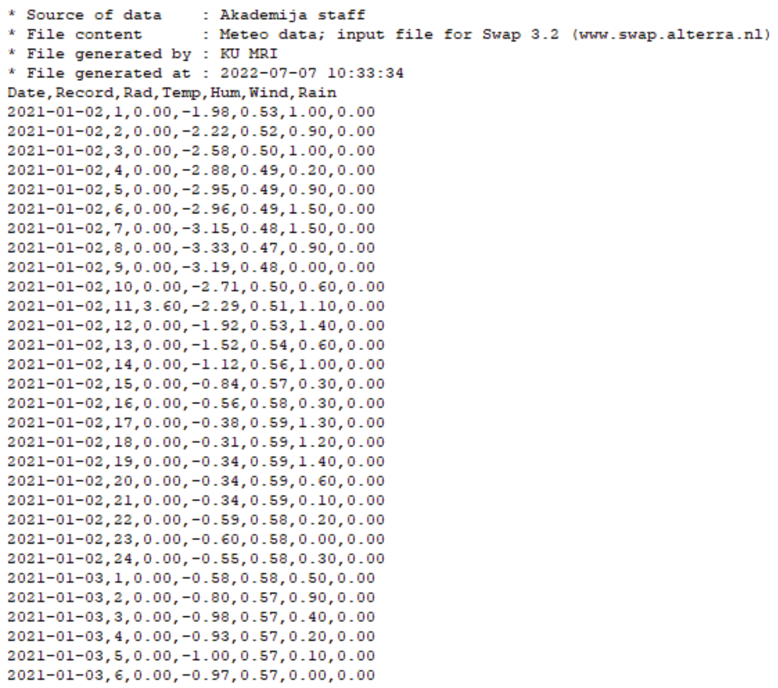
version. The use of Rmarkdown or Rshiny to provide a GUI is being considered. The tool will be upgraded in the OPTAIN cloud under the following path:
*WPs & Tasks/WP3/Scripts and tools/App3_SWAP_soft_calibration_and_visualisation. r*

*Table 4. Summary of the script/tool for soft-calibration and visualisation for field-scale models*

| | |
|---|---|
| **Source data structure:** | SWAP output text files *.vap, *.wba |
| **Processed data structure:** | *.png; figures visualising the modelled and measured data and goodness-of-fit statistics |
| **In use for CS:** | Switzerland (CS 2), Hungary (CS 3, 11), Poland (CS 4), Lithuania (CS 8), Norway (CS 10), Czech Rep. (CS 12) |
| **Language or tool:** | R (v 2021.09.1) |
| **Script availability:** | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| **Appendix no.** | 3 |

## 2.1.4. Sub-daily meteorological data format

Within the SWAP model input, the meteorological records can also be specified with short, constant time intervals (> 15 mins). In this input data format (Figure 8), the solar radiation and rainfall indicate cumulative amounts during the time interval, while air temperature, air humidity, and wind speed denote average values during the time interval (Kroes et al., 2008). Missing data values should be represented with a flag of -99.



```
* Source of data    : Akademija staff
* File content      : Meteo data; input file for Swap 3.2 (www.swap.alterra.nl)
* File generated by : KU MRI
* File generated at : 2022-07-07 10:33:34
Date,Record,Rad,Temp,Hum,Wind,Rain
2021-01-02,1,0.00,-1.98,0.53,1.00,0.00
2021-01-02,2,0.00,-2.22,0.52,0.90,0.00
2021-01-02,3,0.00,-2.58,0.50,1.00,0.00
2021-01-02,4,0.00,-2.88,0.49,0.20,0.00
2021-01-02,5,0.00,-2.95,0.49,0.90,0.00
2021-01-02,6,0.00,-2.96,0.49,1.50,0.00
2021-01-02,7,0.00,-3.15,0.48,1.50,0.00
2021-01-02,8,0.00,-3.33,0.47,0.90,0.00
2021-01-02,9,0.00,-3.19,0.48,0.00,0.00
2021-01-02,10,0.00,-2.71,0.50,0.60,0.00
2021-01-02,11,3.60,-2.29,0.51,1.10,0.00
2021-01-02,12,0.00,-1.92,0.53,1.40,0.00
2021-01-02,13,0.00,-1.52,0.54,0.60,0.00
2021-01-02,14,0.00,-1.12,0.56,1.00,0.00
2021-01-02,15,0.00,-0.84,0.57,0.30,0.00
2021-01-02,16,0.00,-0.56,0.58,0.30,0.00
2021-01-02,17,0.00,-0.38,0.59,1.30,0.00
2021-01-02,18,0.00,-0.31,0.59,1.20,0.00
2021-01-02,19,0.00,-0.34,0.59,1.40,0.00
2021-01-02,20,0.00,-0.34,0.59,0.60,0.00
2021-01-02,21,0.00,-0.34,0.59,0.10,0.00
2021-01-02,22,0.00,-0.59,0.58,0.20,0.00
2021-01-02,23,0.00,-0.60,0.58,0.00,0.00
2021-01-02,24,0.00,-0.55,0.58,0.30,0.00
2021-01-03,1,0.00,-0.58,0.58,0.50,0.00
2021-01-03,2,0.00,-0.80,0.57,0.90,0.00
2021-01-03,3,0.00,-0.98,0.57,0.40,0.00
2021-01-03,4,0.00,-0.93,0.57,0.20,0.00
2021-01-03,5,0.00,-1.00,0.57,0.10,0.00
2021-01-03,6,0.00,-0.97,0.57,0.00,0.00
```

*Figure 8. Text format of sub-daily weather records required for the SWAP model. All lines starting with a "*" character are optional comment lines, not read by SWAP.*

A tool was developed using Python that can produce such an input file from a pre-defined csv format. The original csv file is a direct output from the meteorological station recording system, which records weather information every hour from a local field in a case study site (our case – Lithuania, Dotnuvele river basin). The script is used for creating the SWAP model sub-daily weather information from these specific weather records.

The script can also be adjusted to meet other demands, i.e. to reformat from other specific file formats to SWAP-specific sub-daily files.

*Table 5. Summary of the script for creating and formatting sub-daily meteorological data*

| | |
|---|---|
| Source data structure: | Comma-separated values file, with multiple columns |
| Processed data structure: | Text file, formatted in accordance with |
| In use for CS: | Lithuania (CS 8) |
| Language or tool: | Python (v3.8) |
| Script availability: | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 4 |

### 2.1.5. Autocalibration tool

An autocalibration tool automates the process of model calibration by communicating with the model's inputs and outputs. Such a tool facilitates a large number (e.g. thousands) of model runs using systematically changed alternative parameterizations, and automates the evaluation of those model results against reference data and the identification of the best parameter set. The use of such a tool makes work more efficient, helps harmonize the workflow at each case study, and to document the parameter-space that has been tested. For instance, PEST is a model-independent software that can be used for parameter estimation, sensitivity and uncertainty analysis. PEST stands for "parameter estimation". Given its model-independent features, PEST can be used with any other model whose input and output files are text files. The software comprises two global optimizers, a basic sensitivity analyser, a utility facilitating parallel run management and other utility support programs. More details on PEST are covered by Doherty (2015), including the theory embodied in PEST and its utility support software.

PEST runs the SWAP model with an initial guess of the parameters, compares the model results with observations, adjusts selected parameters using an optimization algorithm, and runs the model again. The procedure of adjusting selected parameters continues until the difference between the model results and observations meets predefined criteria by user. PEST interacts with a model through the model's own input and output files. The general workflow of PEST and interactions with the SWAP model are illustrated in Figure 9.

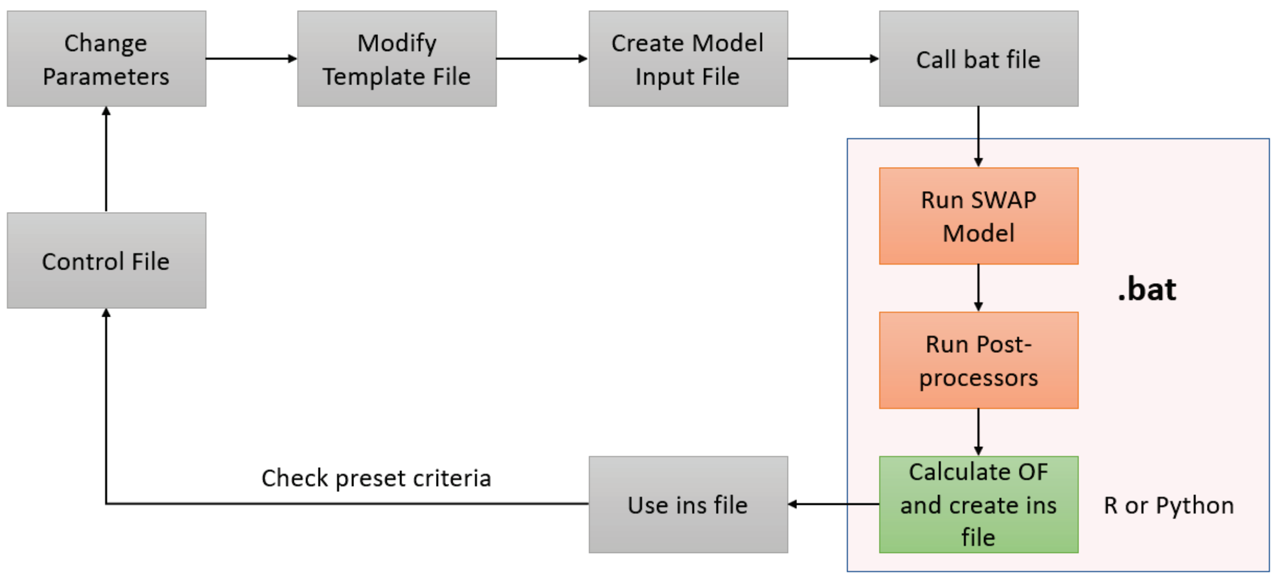


*Figure 9. General workflow of PEST and interactions with SWAP model. OF = Objective Function. Ins file = instruction file*

Because PEST operates in a model-independent manner, it interacts with a model through the model's own input and output files. Hence, no programming is required to use PEST to calibrate a model. However, two scripts have been developed for SWAP-PEST integration to facilitate the autocalibration process in the project.

In the course of parameter estimation or computing sensitivities of model outputs to parameters, PEST must run a model many times. It does this through a call to the operating system. Hence the model must be accessible to a user (and therefore to PEST) through the command line. PEST requires an executable file which can be called from a batch file for consecutive operations in calibration. The batch file is available in Appendix 5.

The objective function (OF), which is used for model calibration and performance analysis, should be calculated after each model run during the calibration process. To calculate the OF, a script (R and Python versions) that reads both SWAP model output and measured data is developed. The script (Python version) is available in Appendix 6. This script must be called after each run; hence it must be defined in the batch file. SWAP and PEST have been integrated successfully and all required PEST files (control, template and instructions files) and introductory documentation together with SWAP files have been provided in the OPTAIN cloud under the following path:
*WPs & Tasks/WP3/Scripts and tools/SWAP - PEST Autocalibration/*

*Table 6. Summary of the script/tool for autocalibration of field-scale models*

| | |
|---|---|
| Source data structure: | Default SWAP model text files |
| Processed data structure: | Modified SWAP model input text files |
| In use for CS: | Switzerland (CS 2), Hungary (CS 3, 11), Poland (CS 4), Lithuania (CS 8), Norway (CS 10), Czech Rep. (CS 12) |
| Language or tool: | Python (v3.9), R (v 2021.09.1), PEST |
| Script availability: | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 5, 6 |

## 2.2. Catchment-scale model (SWAT)

For the catchment-scale, OPTAIN uses the integrated model SWAT+ (Bieger et al., 2017), which is the newest version of the acknowledged Soil and Water Assessment Tool (SWAT; Arnold et al., 2012). SWAT+ is used to quantify the effects of water management measures on crop production and environment under changing climate, but it is also capable to simulate many NSWRM (e.g. crop rotation, conservation tillage, filter strips, retention ponds, tile drains). Thanks to the new and more flexible routing structure considering landscape positions of Hydrological Response Units (HRUs), SWAT+ is able to account for topological filter and retention effects in the routing of water, nutrients and sediments between upland and floodplain HRUs (e.g. useful to simulate riparian buffers). The model uses several inputs such as climate, topography, land use and management and soil data. The following section presents the procedures that we used to process and harmonize the necessary input data.

### 2.2.1. Future meteorological data

Data for future meteorological scenarios were needed in OPTAIN in order to be able to run scenario analyses about the environmental and societal benefits of currently used or implementable NSRWMs. Taks 3.1 of OPTAIN developed the required datasets (Honzak & Pogačar, 2022) and provided these in NetCDF format based on the ERA5 Land climate data set. The climate variables are provided in a gridded format. For the catchment scale modelling with SWAT+ weather stations have to be assigned to the spatial units of a model setup. Depending on the spatial resolution of the gridded meteorological data and the spatial domain of the SWAT model setup, it can be necessary to aggregate the meteorological data for the spatial units of a SWAT+ model setup.

*Table 7. Summary of the script for pre-processing future meteorological data of the catchment-scale model*

| | |
|---|---|
| **Source data structure:** | NetCDF |
| **Processed data structure:** | Text files for SWAT+ model input |
| **In use for CS:** | All case studies |
| **Language(-s) or tool (-s):** | R (v 4.2.1/ 2021.09.1), aRastoCAT package |
| **Script availability:** | Available for public use at: https://github.com/chrisschuerz/aRastoCAT |

The R package 'aRastoCAT – a Raster Climate data Aggregation Tool' (Schürz, 2020) was developed to aggregate gridded climate data for spatial units of a SWAT model setup and generate the required weather time series input files for a model setup. Figure 10 illustrates the workflow for the weather data aggregation. The gridded weather data is organized as a 3-dimensional array in the NetCDF file where the first two dimensions are the latitude and longitude of the spatial array and the third dimension gives the time steps. To spatially aggregate the climate data to the spatial model units the grid cells of the weather data are intersected with the polygons of the spatial units of the SWAT model setup. In the following, weights for each cell and for each spatial unit of the model setup are calculated. Based on the area weights, weighted mean average values of the climate variable are calculated for each spatial unit and each time step. This routine can be performed in R with the aRastoCAT function 'aggregate_ncdf()'. The function returns a table with column-wise time series of a climate variable for each spatial unit (see Figure 11). The routine is performed with all climate variables that should be included in a SWAT model setup. Based on the climate variable time series tables SWAT weather inputs can be generated with the aRastoCAT R function ‘write_SWATweather()’. For OPTAIN a small template R script for weather data preparation will be prepared and shared with the OPTAIN case study leads.

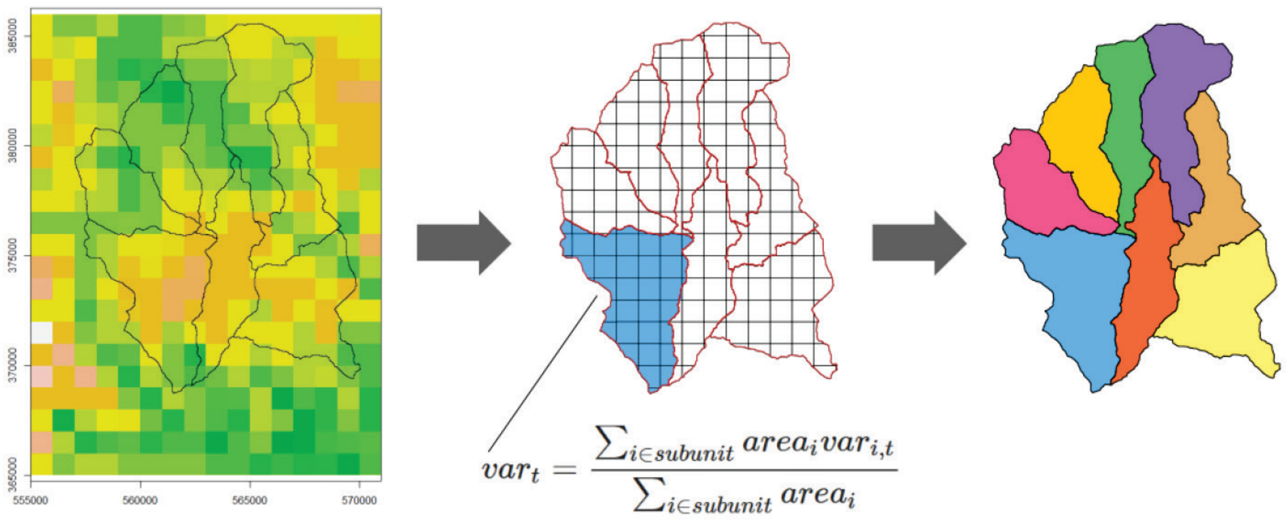
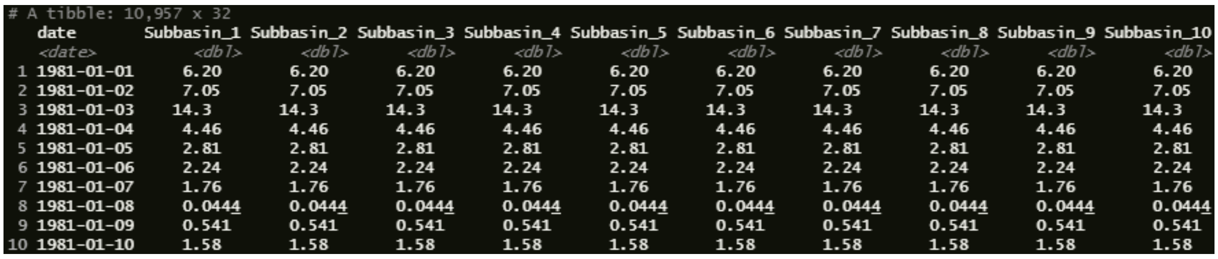

385000
380000
375000
370000
365000
555000
560000
565000
570000
$$var_t = \frac{\sum_{i \in subunit} area_i var_{i,t}}{\sum_{i \in subunit} area_i}$$

*Figure 10.* Aggregation workflow with aRastoCAT; weather data arrays (here shown one timestep for the spatial grid extent) are aggregated by intersecting the grid cells with the spatial units of a model setup (here subbasins) and calculating the area weighted averages.



```
# A tibble: 10,957 x 32
   date       Subbasin_1 Subbasin_2 Subbasin_3 Subbasin_4 Subbasin_5 Subbasin_6 Subbasin_7 Subbasin_8 Subbasin_9 Subbasin_10
   <date>          <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>       <dbl>
 1 1981-01-01     6.20       6.20       6.20       6.20       6.20       6.20       6.20       6.20       6.20        6.20
 2 1981-01-02     7.05       7.05       7.05       7.05       7.05       7.05       7.05       7.05       7.05        7.05
 3 1981-01-03    14.3       14.3       14.3       14.3       14.3       14.3       14.3       14.3       14.3        14.3
 4 1981-01-04     4.46       4.46       4.46       4.46       4.46       4.46       4.46       4.46       4.46        4.46
 5 1981-01-05     2.81       2.81       2.81       2.81       2.81       2.81       2.81       2.81       2.81        2.81
 6 1981-01-06     2.24       2.24       2.24       2.24       2.24       2.24       2.24       2.24       2.24        2.24
 7 1981-01-07     1.76       1.76       1.76       1.76       1.76       1.76       1.76       1.76       1.76        1.76
 8 1981-01-08     0.0444     0.0444     0.0444     0.0444     0.0444     0.0444     0.0444     0.0444     0.0444      0.0444
 9 1981-01-09     0.541      0.541      0.541      0.541      0.541      0.541      0.541      0.541      0.541       0.541
10 1981-01-10     1.58       1.58       1.58       1.58       1.58       1.58       1.58       1.58       1.58        1.58
```

*Figure 11.* Example for aggregated NetCDF precipitation data for the subbasins of a SWAT model setup. In this example all timeseries would be identical due to a small model domain and a coarse resolution of the gridded weather data

### 2.2.2. Soil physical and hydrological properties

SWAT+ requires information on soil physical and hydraulic properties that are essential for the model to calculate water and solute fluxes and storage capacities. In OPTAINs deliverable D3.2 (Szabó et al., 2022) it was described how these properties could be derived if these are not available for part of or the entire CS. We provided an R script (pred_soil_prop.R) for the calculation of moist soil bulk density, available water capacity, saturated hydraulic conductivity, moist albedo and USLE soil erodibility factor.

*Table 8. Summary of script for Soil physical and hydrological properties*

| | |
|---|---|
| Source data structure: | CSV |
| Processed data structure: | Text files for SWAT+ model input |
| In use for CS: | Felső-Válicka (CS 3b), all other case studies |
| Language(-s) or tool (-s): | R (v 4.1.2), euptfv2 R package |
| Script availability: | Available for public use at: https://doi.org/10.5281/zenodo.6684582 |
| Appendix no. | 7 |

The R script uses the soil property table formatted as required by the SWAT+ and inserts the missing soil properties into it. For the computation of soil hydraulic properties, the euptfv2 R package (Szabó et al., 2021; Weber et al., 2020) is used. The calculation of the other soil physical properties is based on Wessolek et al. (2009), Abbaspour et al. (2019), and Sharpley & Williams (1990).

### 2.2.3. Soil phosphorus content

The SWAT+ model uses soil labile phosphorus (P) content of the surface soil layer to initialize the different P pools. In most cases soil labile P is not available because its measurement is time consuming. Instead, plant available soil P content is the form which is frequently analysed for practical reasons. However, data on plant available soil P content is often not available for entire catchment areas, therefore a method was provided to derive soil P content of the surface layer. One of the most commonly used P test method is the Olsen method, also the LUCAS Topsoil Survey dataset contains soil P content of 20,000 soil samples in Europe measured by this method. Thus, we decided to map soil Olsen-P content for data scarce areas based on the LUCAS Topsoil Survey dataset.

We provided a uniform method for producing Olsen-P maps for any of the OPTAIN CSs. The computation needs the LUCAS Topsoil Survey dataset, LUCAS survey database, local land use or land cover map and European agro-climate zone map, all are freely accessible. The detailed explanation of the workflow is included in D3.2 (Szabó et al., 2022). Figure 12 shows the soil P content maps derived for CS3b.

*Table 9. Summary of the script for deriving soil phosphorus content*

| | |
|---|---|
| Source data structure: | CSV, XLS, GeoTIFF and SHP |
| Processed data structure: | GIS raster file |
| In use for CS: | Csorsza (CS 3a), Felső-Válicka (CS 3b), Tetves (CS 11) |
| Language(-s) or tool (-s): | R (v 4.1.2) |
| Script availability: | Available for public use at: https://doi.org/10.5281/zenodo.6656537 |
| Appendix no. | 8, 9 |

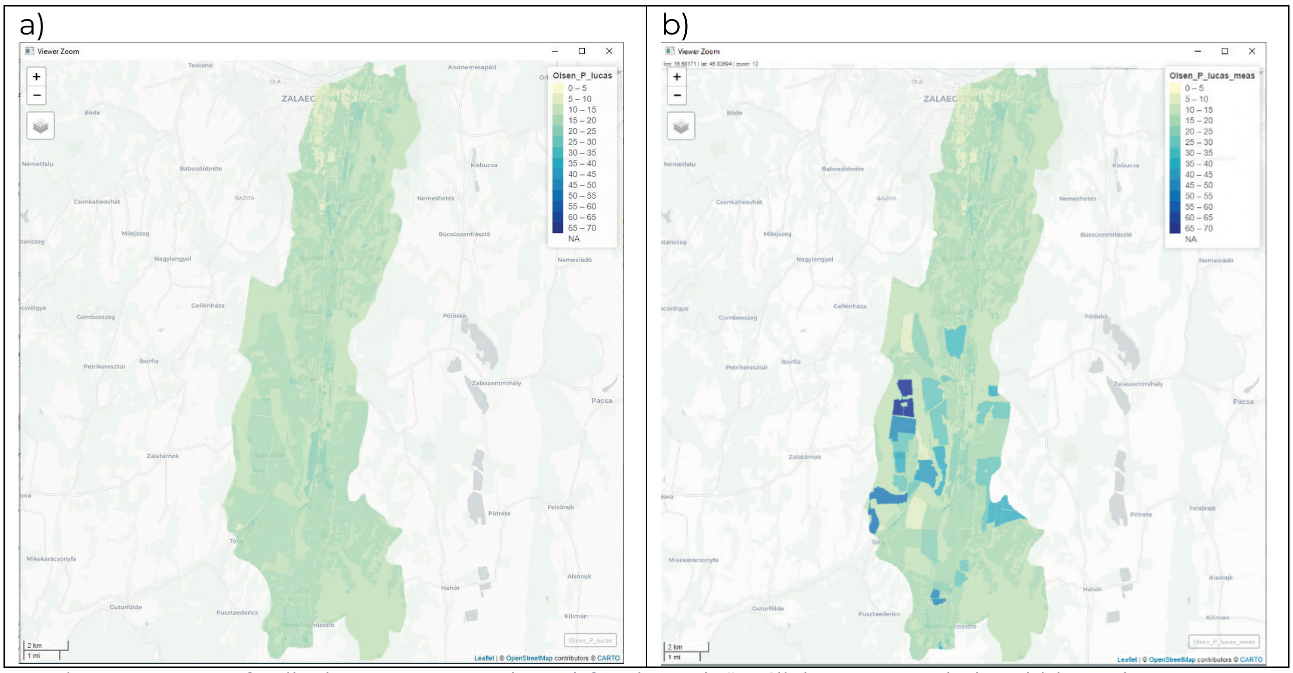


*Figure 12. Map of soil Olsen-P content (ppm) for the Felső-Válicka case study (CS3b) based on LUCAS Topsoil data and local land use map (20 m resolution) a) without and b) with locally measured soil P content.*

For the preparation of nutrients.sol input file for the SWAT+ model, a script was provided to compute geometric mean of the soil Olsen-P content by Hydrologic Response Units (HRUs). The script – meanP_by_HRU.R – is available in Appendix 9 and from the OPTAIN cloud under the following path:
*/ WPs & Tasks/WP3/Scripts and tools/App9_Add_soilP_content_to_HRU.r*

### 2.2.4. Crop map pre-processing

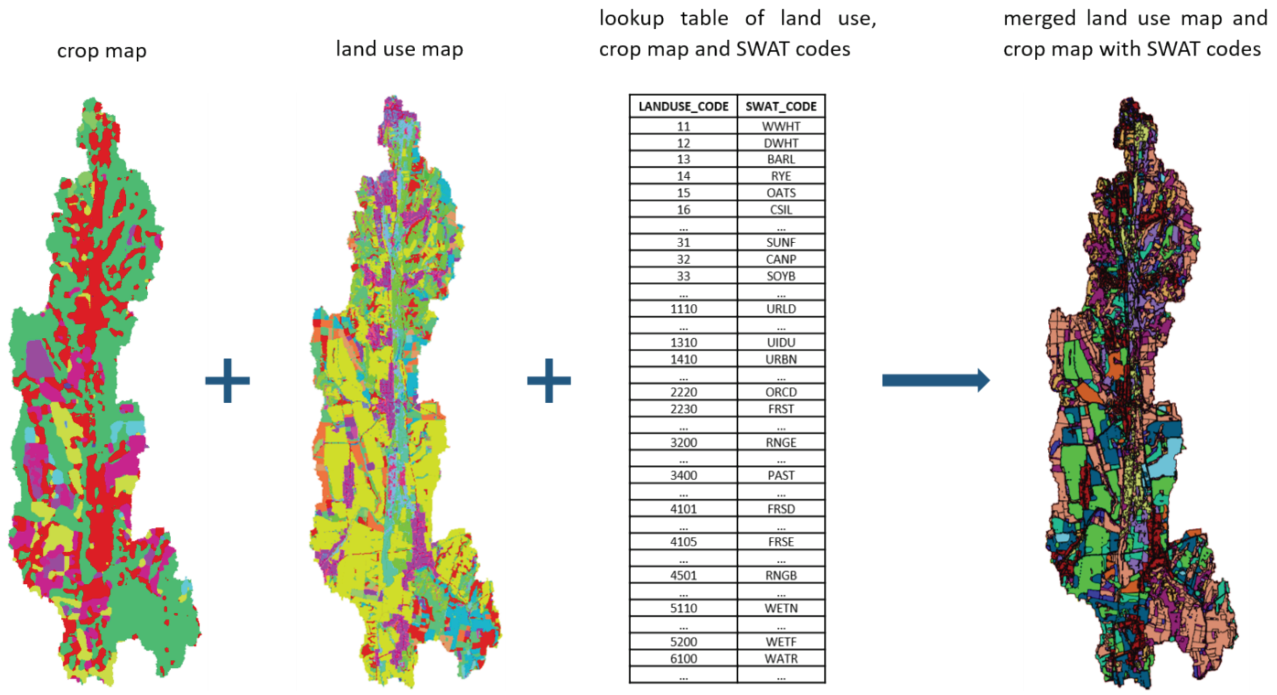
Information on crop types are required for the SWAT+ model. The crop maps are used to further specify agricultural land categories of the land-use map and the SWAT model (AGRL, AGRR, AGRC) (Neitsch et al., 2009). The crop map is a fundamental data to define land use management and crop yield. Under the land use management input files of SWAT+, it is needed to provide information about characteristic soil management, fertilization, planting and harvesting schedule. These data are defined specifically in the decision table. The time series crop maps provide information to specify crop rotation. The knowledge of crop type is important for defining related existing and possible NSWRM as well.

If crop maps are not available for the entire modelling period and/or case study, those can be derived from remote sensing data. A crop classification approach has been derived, which predict crop types based on time series reflectance data of Sentinel-1A and -1B satellite radar images with random forest method. The approach has been derived in Google Earth Engine Platform. A detailed description of the workflow is available in the D3.2 deliverable (Szabó et al., 2022). The Google Earth Engine script is accessible on ZENODO (https://doi.org/10.5281/zenodo.6669643) and Appendix 10. An R script has been provided for CS leaders to merge the local land use map with the derived crop maps (Figure 13.), which is available from Appendix 11. Required inputs are:

- a table which shows how to link the merged land use map categories (defined based on LUCAS documentation and CORINE terminology) with the SWAT+ crop type codes,
- the crop map derived in Google Earth Engine Platform with the crop classification script,
- the land use map of the catchment.

*Table 10. Summary of the script for generating crop maps as input to catchment-scale models*

| | |
|---|---|
| Source data structure: | GeoTIFF, SHP, CSV |
| Processed data structure: | SHP |
| In use for CS: | Csorsza (CS 3a), Felső-Válicka (CS 3b), Tetves (CS 11) |
| Language(-s) or tool (-s): | Google Earth Engine, R (v 4.1.2) |
| Script availability: | Available at for public use: [https://doi.org/10.5281/zenodo.6669643](https://doi.org/10.5281/zenodo.6669643) And OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 10, 11 |


crop map
land use map
lookup table of land use, crop map and SWAT codes
merged land use map and crop map with SWAT codes

| LANDUSE_CODE | SWAT_CODE |
|---|---|
| 11 | WWHT |
| 12 | DWHT |
| 13 | BARL |
| 14 | RYE |
| 15 | OATS |
| 16 | CSIL |
| ... | ... |
| 31 | SUNF |
| 32 | CANP |
| 33 | SOYB |
| ... | ... |
| 1110 | URLD |
| ... | ... |
| 1310 | UIDU |
| 1410 | URBN |
| ... | ... |
| 2220 | ORCD |
| 2230 | FRST |
| ... | ... |
| 3200 | RNGE |
| ... | ... |
| 3400 | PAST |
| ... | ... |
| 4101 | FRSD |
| ... | ... |
| 4105 | FRSE |
| ... | ... |
| 4501 | RNGB |
| ... | ... |
| 5110 | WETN |
| ... | ... |
| 5200 | WETF |
| 6100 | WATR |
| ... | ... |

*Figure 13. Workflow of merging land use map with crop map and assigning SWAT land use codes.*

## 2.3. Metadata for model inputs

Following the **INfrastructure for SPatial InfoRmation in Europe (INSPIRE)** initiative ([https://inspire.ec.europa.eu/](https://inspire.ec.europa.eu/)), the assembling and sharing of the datasets prepared for the implementation of the modelling studies of the OPTIAN project will be accomplished by creating their corresponding metadata files based on the non-binding technical guidelines for implementing dataset and service metadata based on ISO/TS 19139:2007 document ([TG INSPIRE (europa.eu)](TG INSPIRE (europa.eu))), version 2.0.1). The metadata of the prepared model input datasets will be created using a simple Excel form spreadsheet, which was initially developed within the GeoSmartCity project and adapted for the OPTAIN project use by the teams from Centre for Agricultural Research, HU, and Klaipeda University, LT (Čerkasova et al., 2021). The metadata files can be generated one by one (spreadsheet-based approach), or in bulk using a custom R script (script-based approach).

### 2.3.1. Spreadsheet-based approach

The simple Excel spreadsheet-based tool to collect INSPIRE XML19139 (TG INSPIRE (europa.eu)) compliant metadata of model input datasets can be found in in the OPTAIN cloud under the following path:
*WPs & Tasks|WP3|Task_3_1|templates|Metadata_script|*

This directory holds a complete user manual, the necessary tools, and example files. The Excel form tool itself has separate sheets describing the tool (Readme), the necessary form sheets for metadata compilation (Form and Form_R), and the authorship (Credits).

The spreadsheet-based approach, or a single metadata file compilation approach, offers a simple file generation option with a straightforward press-one-button technique. The form sheet, where the required information about a single dataset has to be specified, has self-explanatory instructions on what information is needed with additional comments in cells with red corner indicator, which becomes visible when you hover your cursor over the cell. Some cells in the form can be edited as free text, others contain values taken from dropdown list, and some contain already predefined information. After filling out the form, user has to press the "Export XML" cell and specify the location for saving the output xml file.

*Table 11. Summary of the script for the generation of metadata for model inputs*

| | |
|---|---|
| **Source data structure:** | Comma-separated values file, with multiple columns |
| **Processed data structure:** | XML metadata file |
| **Used for CS:** | All Case Studies |
| **Language(-s) or tool (-s):** | VBA embedded in MS Excel 2019 |
| **Script availability:** | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |

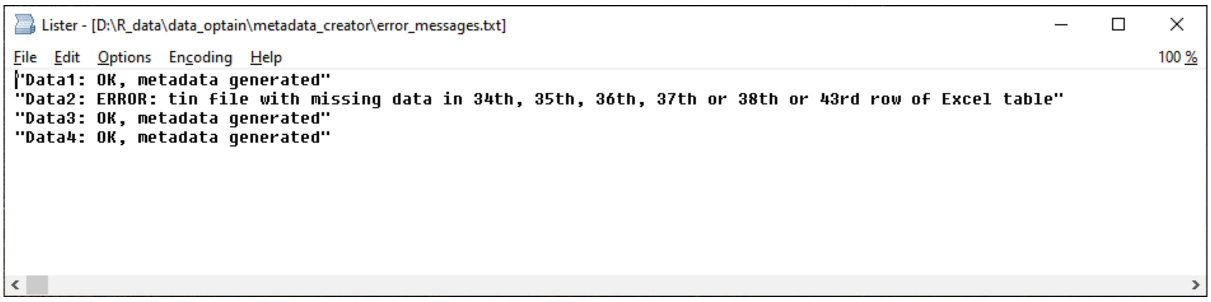
### 2.3.2. Script-based approach

For the creation of multiple metadata XML files, we provided an R script, metadata_creator.R., which was written by using the geometa R package (CRAN - Package geometa (r-project.org); Blondel, 2020). In this approach the Form_R sheet of the Metadata_form.xlsm file has to be filled in. This sheet is almost identical with the Form sheet, the differences are that i) the metadata information of multiple datasets can be provided in separate columns, ii) some information is not obligatory because it is generated by the R script, e.g.: Metadata date stamp, Geographic bounding box, Spatial resolution – for raster files, etc.). Absolute file path has to be given.

*Table 12. Summary of the script for generating script-based metadata for model inputs*

| | |
|---|---|
| Source data structure: | Comma-separated values file, with multiple columns |
| Processed data structure: | XML metadata file |
| Used for CS: | All Case Studies |
| Language(-s) or tool (-s): | R (v 4.1.2) |
| Script availability: | OPTAIN cloud (Used internally by OPTAIN partners, available on demand) |
| Appendix no. | 12 |

The derived XML metadata files are written into the folder of the original files. A logfile (error_messages.txt) is also generated in the same folder where the R script file was saved. It includes the errors of creating the XML files if there are any of them (Figure 14).


Lister - [D:\R_data\data_optain\metadata_creator\error_messages.txt]
File Edit Options Encoding Help
100 %

```
"Data1: OK, metadata generated"
"Data2: ERROR: tin file with missing data in 34th, 35th, 36th, 37th or 38th or 43rd row of Excel table"
"Data3: OK, metadata generated"
"Data4: OK, metadata generated"
```

*Figure 14. An example for the content of the error_messages.txt file.*

The script is available from Appendix 12 and in the OPTAIN cloud under the following path: *WPs & Tasks|WP3|Task_3_1|templates|Metadata_script|*

# References

Abbaspour, K.C., AshrafVaghefi, S., Yang, H. & Srinivasan, R. 2019. Global soil, landuse, evapotranspiration, historical and future weather databases for SWAT Applications. Scientific Data, 6:263.

Arnold, J.G., Moriasi, D. N., Gassman, P. W., Abbaspour, K. C., White, M. J., Srinivasan, R., Santhi, C., Harmel, R. D., van Griensven, A., Van Liew, M. W., Kannan, N. & Jha, M. K. (2012): SWAT: Model Use, Calibration, and Validation. Transactions of the ASABE, Vol. 55(4): 1491-1508.

Bieger, K., Arnold, J.G., Rathjens, H., White, M.J., Bosch, D.D., Allen, P.M., Volk, M. & Srinivasan, R. (2017): Introduction to SWAT+, a completely restructured version of the Soil and Water Assessment Tool. *Journal of the American Water Resources Association* 53(1): 115-130.

Blondel E. 2020. geometa: Tools for Reading and Writing ISO/OGC Geographic Metadata. R package version 0.6-3

Čerkasova, N., Idzelytė, R., Banovec, P. & Glavan, M. (2021): *Common working environment with standardised metadata for the harmonised reporting of project outputs*. Deliverable D6.1 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756

Honzak, L. & Pogačar, T. (2022): *Climate scenarios for integrated modelling*. Deliverable D3.1 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756

Kroes, J.K., Van Dam, J.C., Groenendijk, P., Hendriks, R.F.A. & Jacobs, C.M.J. (2008): SWAP version 3.2. Theory description and user manual. Wageningen, Alterra, Alterra

Kroes, J. G., et al. "SWAP version 4; Theory description and user manual, Report 2780, Wageningen Environmental Research, Wageningen." (2017).

Leib, B.G., Jabro, J.D., & Matthews, G.R. 2003. Field evaluation and performance comparison of soil moisture sensors. Soil Science, 168(6):396-408.

Mittelbach, H., Lehncer, I., & Senerviratne, S.I. 2012. Comparison of four soil moisture sensor types under field conditions in Switzerland. Journal of Hydrology, 430-431:39-49.
Moriasi, D.N., Gitau, M.W., Pai, N. & Daggupati, P. 2015. Hydrologic and water quality models: performance measures and evaluation criteria. Transactions of the ASABE, 58(6):1763-1785.

Neitsch, S.L., Arnold, J.G., Kiniry, J.R. & Williams, J.R. 2009. Soil and Water Assessment Tool Theoretical Documentation—Version 2009. 618.

Nash, J.E., & J.V. Sutcliffe. 1970. River flow forecasting through conceptual models: Part 1. A discussion of principles. J. Hydrology 10(3): 282-290.

Szabó, B., Mészáros, J., Kassai, P., Braun, P., Nemes, A., Farkas, Cs., Čerkasova, N., Monaco, F., Chiaradia, E. A., Witting, F. (2022): Solutions to overcome data scarcity. Deliverable D3.2 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756

Schürz, C. (2020): aRastoCAT: a Raster Calimate data Aggregation Tool, R package version 0.2.2, available at: https://github.com/chrisschuerz/aRastoCAT

Sharpley, A.N. & Williams, J.R. 1990. EPIC — Erosion / Productivity Impact Calculator: 1. Model Documentation.

Szabó, B., Mészáros, J., Kassai, P., Braun, P., Nemes, A., Farkas, C., Čerkasova, N., Monaco, F., Chiaradia, E.A. & Witing, F. 2022. Solutions to overcome data scarcity. Deliverable D3.2 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756.

Szabó, B., Weynants, M. & Weber, T.K. 2021. Updated European Hydraulic Pedotransfer Functions with Communicated Uncertainties in the Predicted Variables (euptfv2). Geosci. Model Dev., 14, 151–175, (At: https://doi.org/10.5194/gmd-14-151-2021. ).

Weber, T.K.D., Weynants, M. & Szabó, B. 2020. R package of updated European hydraulic pedotransfer functions (euptf2). (At: https://doi.org/10.5281/zenodo.4281045. ).

Wessolek, G., Kaupenjohann, M. & Renger, M. 2009. Bodenphysikalische Kennwerte und Berechnungsverfahren für die Praxis. Bodenökologie und Bodengenese, 40, 1–80, (At: https://www.boden.tu-berlin.de/fileadmin/fg77/_pdf/Rote_Liste/Rote_Reihe_Heft_40.pdf. ).

Witing, F. & Volk, M. 2021. *OPTAIN Knowledge management plan*. Milestone MS24 EU Horizon 2020 OPTAIN project, Grant agreement No. 862756

# Appendixes

# 1. Script for creating meteorological input files for SWAP in daily resolution

```
#############################################################
# Script for converting Excel meteorological data template into SWAP demanded
# meteo input data format
# Author: Moritz Shore | moritzshore@gmail.com | NIBIO | SWAP
# Date: 14.02.2022
#############################################################

# !!! #
# There are still some active issues that need to be considered:
  # 1. The null/NA value for Precipitation -- What can it be? SWAP wants -99.0, BUT
  #    SWAP will not accept negative values for precipitation... so what to do???
  # 2. The units of some of the input variables are still unconfirmed:
      # --> WET (d) --> presumably fraction of day in which it rains [between 0 and 1]..
unsure!

# Libraries
library(readxl) # for excel sheet import
library(stringr) # for string manipulation

# select file
fname <- file.choose()
# select outpath
outpath <- choose.dir(default = "", caption = "Select path for output")

# read in source excel
meteo_excel <- read_excel(fname)

# read in data
outdf <- data.frame(Station = meteo_excel$Station,
                # if you change the date format of the excel template,
                # make sure to check these date lines of code too
                DD = format(as.Date(meteo_excel$date, format = "%Y-%m-%d"), format =
"%d"), # date formatting
                MM = format(as.Date(meteo_excel$date, format = "%Y-%m-%d"), format =
"%m"), # date formatting
                YYYY = format(as.Date(meteo_excel$date, format = "%Y-%m-%d"), format =
"%Y"), # date formatting
                RAD = meteo_excel$RAD, #(kJ/m2)
                Tmin = meteo_excel$T_MIN, #(oC)
                Tmax = meteo_excel$T_MAX, #(oC)
                HUM = meteo_excel$HUM, #(kPa)
                WIND = meteo_excel$WIND, #(m/s)
                RAIN = meteo_excel$P, #(mm)
                ETref = meteo_excel$ETref, #(mm)
                WET = meteo_excel$WET #(d)
```

```
          )

# save file name before reformatting to swap style
# this can be customized, if desired. the only important thing is that the
# swap main file recognizes it, and that the file extension is not changed!
filename = outdf$Station[1]

# reformat station name (this may be unecessary --> untested)
outdf$Station  = paste0("'", filename, "'")

# reformatting data to swap's liking #

# replace null [ISSUE: BANDAID FIX]
# it really shouldn't be zero, but what can I change it to??
# -99.0 is Swap's "error value / NA / NULL" but it wont let RAIN and possibly
# other columns be be negative...
# if your NULL/NA values are different, please update this line!
outdf[outdf == "NULL"] <- 0.0
outdf[2:12] <- sapply(outdf[2:12],as.numeric) # convert to numeric

# decimal reformatting [C-style string formatting!]
# the number refers to the number of decimal spaces
outdf$RAD = sprintf("%.1f", outdf$RAD)
outdf$Tmin = sprintf("%.1f", outdf$Tmin)
outdf$Tmax = sprintf("%.1f", outdf$Tmax)
outdf$HUM = sprintf("%.2f", outdf$HUM)
outdf$WIND = sprintf("%.1f", outdf$WIND)
outdf$RAIN = sprintf("%.1f", outdf$RAIN)
outdf$ETref = sprintf("%.1f", outdf$ETref)
outdf$WET = sprintf("%.4f", outdf$WET)

# Split by year and write to file
# SWAP requires meteo-input files seperate per year with a silly file extension
# of .XXX where xxx are the last 3 digits of the year...
years = c(min(outdf$YYYY):max(outdf$YYYY)) # range of years
for (year in years) {
  # generate swap's silly file extension
  file_ext = paste0(".",substr(year, 2,4))
  # write the file
  write.table(outdf[which(outdf$YYYY == year),], paste0(outpath, "\\", filename, file_ext),
quote = F, row.names=F, sep=",")
  print(paste0(filename, file_ext))
}

cat(paste0("\nDone! Files (",length(years),") written to ", outpath,"\n"))
```

# 2. Script for reference data quality check

```
#################################################
## REFERENCE DATA QUALITY CHECK TOOL – MAIN.R ##
#################################################
##Designd to prepare soil water reference data for the SWAP model
##Loading, cleaning, visualizing results, writing to SWAP calibration format
##Input data format requirements: see example file!
##For any questions, comments, errors, suggestions please contact Levente Czelnai
czelnai.levente@atk.hu
##Last edited on 2022-07-12
#################################################
##Special attention needed by the user in the input data:
##Sites names in the 'Sites' sheet must be the same as the name of other worksheets in
the excel file!
##Colnames can be different from the example excel file except for 'date'!
##The 'date' column MUST be a date column format, NOT a text variable!
#################################################
##User needs to modify:
##/main.R/line 60-61: Mandatory (User needs to specify if he wants to run the script from
raw data or continue with cleaned datasets!)
##      /line 56-57: Mandatory (User needs to specify his input data parameters!)
##/load.R/line 10:    Mandatory (User needs to give the input data name!)
##      /line 79:    Optional (Use it if soil water content is in %!)
##/clean_manual.R/line 59-82:  Optional (Use your own site name and depth acronyms if
you want to perturb the wilting point and saturated water content!)
##             /line 214-237:Optional
##             /line 87/242: Optional
#################################################
##Attention !!! There are still some active issues that need to be considered:
##Clean1.R is not ready yet, so take into consideration that your data needs to be filtered
out of extreme values, missing records should be filled with NAs, etc!
#################################################

##Setting workplace & paths:
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

folder <- print(getwd())
input_path <- "./input/"
output_path <- "./output/"
file_list <- list.files(path=paste0(folder, "/input"), pattern="*xlsx")

cat(
  "\n====[Setting                                    workplace                                    &
paths]=================================================================
======\n",
  "Working directory:\n",
  print(getwd()), "\n",

  "Input files:\n",
  print(file_list),"\n"
```

```
)
##Loading custom function library:
source("function.R", encoding = "UTF-8")

##Installing packages:
using("readxl","readr","validate","plotrix","ggplot2",
      "dplyr","hrbrthemes","viridis","hrbrthemes",
      "xlsx","purrr","stringr","plotly","ggdark",
      "tidyr","naniar","knitr")

##Settings for the use of the scrip!
##Specify input data:
vars<-c("W","T","ta") ##Input data colnames of your soil water content, soil temperature
and average air temperature
site<-c("F","G","A","V") ##Acronyms for your case study sites in your input data
(F=Forest,G=Grassland,A=Arable,V=Vineyard)
##Specify how you want to use the script:
##FALSE if results already available/cleaned and only loading them is needed!
load_results <- F
clean_results <- F
cat(
  "\n====[Settings for the use of the
script!]=================================================================\n",
  "load results:",
  load_results,"\n",
  "clean_results:",
  clean_results,"\n"
)

##Reading data:
if(load_results){
  source("load.R")
  }else{
    df_m <- readRDS(paste0(output_path, "master.RData"))
    load(paste0(output_path, "globenv.RData"))
    cat(
      "\n====[Loading
data]=================================================================
=================\n",
      "Loading data is finished. Results are in df_m masterfile!\n"
      )
}

##Cleaning data:
if(clean_results){
  #source("clean1.R")
  source("clean2.R")
  }else{
    df_c <- readRDS(paste0(output_path, "cleaned2.RData"))
    cat(
    "\n====[Cleaning
data]=====================================================================
================\n",
    "Cleaned data is loaded. Results are in df_c input file!\n"
    )
 }

cat(
 "\n====[Cleaning                                                      phase
II.]======================================================================
===========\n",
 "To continue manual cleaning is needed!\n",
 "Use R script:",
 "clean_manual.R\n"
)

LOAD.R####################################################
##   ######### LOADING DATA ###########
#################################################

cat(
 "\n====[Loading
data]=====================================================================
=================\n"
)

##Reading all files into a list!
input_data     <-     paste0("c:/Users/csfa/Csillan/BOBIKAM/Bp2022/",     "input-csorsza-
demo.xlsx")
sheets <- getSheets(loadWorkbook(input_data))
##Loading van-genuchten parameters for every site:
params <- read.xlsx(input_data, names(sheets)[1], header=TRUE)
##Loading soil water content, soil temperature and atmospheric temperature data for
every site:
ids <- names(sheets)[!names(sheets) %in% "Sites"]
input_lst <- list()
for (id in ids){
 print(paste("Reading site", id, "data."))
 df <- read.xlsx(input_data, id, header=TRUE)
 pars <- names(df)[-1]
 for (par in pars){
   input_lst[[id]][[par]] <- df[,c("Date", par)] %>%
    #drop_na() %>%
    mutate(Date = as.Date(Date, "%Y-%m-%d"))
 }
}
#print("Loading from the input sheets is finished. Input data is stored in input_lst.")

cat(
 "\n...\n"
)
```

```
##Creating a masterfile!
i_lst <- input_lst

df_r1 <- NULL
for (n in names(i_lst)){
  for (p in vars[1]){
    df1 <- i_lst[n][[1]][p][[1]] %>% mutate(Sites = n)
    if (is.null(df_r1)){
      df_r1 <- df1
    } else {
      df_r1 <- bind_rows(df_r1, df1)
    }
  }
}
df_r2 <- NULL
for (n in names(i_lst)){
  for (p in vars[2]){
    df2 <- i_lst[n][[1]][p][[1]] %>% mutate(Sites = n)
    if (is.null(df_r2)){
      df_r2 <- df2
    } else {
      df_r2 <- bind_rows(df_r2, df2)
    }
  }
}
df_r3 <- NULL
for (n in names(i_lst)){
  for (p in vars[3]){
    df3 <- i_lst[n][[1]][p][[1]] %>% mutate(Sites = n)
    if (is.null(df_r3)){
      df_r3 <- df3
    } else {
      df_r3 <- bind_rows(df_r3, df3)
    }
  }
}

df_m = merge(df_r1, df_r2, by=c("Sites","Date"), all=F, sort=F)
df_m = merge(df_m, df_r3,  by=c("Sites","Date"), all=F, sort=F)

##Fixing the column names!
colnames(df_m) <- c("sites","date","swc","temp","ta")
colnames(params) <- c("sites","suc42","tethas")
##Rounding to 1digits!
df_m[3:length(df_m)] <- round(df_m[3:length(df_m)],digits=1)
##Transform swc values from percentage!
df_m[3] <- df_m[3]/100
##Defining THRESHOLDS!
df_m = merge(df_m, params, by=c("sites"), all=F, sort=F)
##Force everything into numeric!
df_m[3:length(df_m)] <- df_m[3:length(df_m)] %>% map(as.numeric)
```

```
##Saving loaded data!
saveRDS(df_m, file = paste0(output_path, "master.RData"))
save(list=c("input_data","site"), file = paste0(output_path, "globenv.Rdata"))

##Cleaning environment a bit!
rm(df,df1,df2,df3,df_r1,df_r2,df_r3,vars,params,input_lst,i_lst,input_path,id,ids,n,p,par,pars,
file_list,folder,sheets)

cat(
  "\nLoading data is finished! Results are in df_m masterfile!",
  "\nThe masterfile is saved in output/master.RData file under working directory!\n",
  View(df_m)

)
```

CLEAN_MANUAL.R

```
#########################################
##   ######### Data Quality Check #########    ##
#################################################
## Guidelines
#################################################
## -Run these commands line by line!
## -You may skip the modifications of the wilting point and saturated water content!
## -If you want to play with setting up the blacklisted df_1 dataset, you must run the lines
50-60 after modifying the multiplier!
## -In line 75-83 you can get the density function of the given sites! It can help to decide
how much you should change the thresholds!
## -In line 87-91 you can save the blacklisted data and select a few to save if you see the
necessity of that.
#################################################
#################################################
##SWCs should be removed below the wilting point (WP (4.2)!
cat(
  "\n====[Manually                                                    cleaning
data]==================================================================
=========",
  "\n4.Soil Water Content should be removed below the wilting point!\n"

)
##Blacklisted data!
df_1 <- subset(df_c,!(swc>suc42))

##Data Overview!
site  <-  df_1  %>%  dplyr::group_by(sites)  %>%  dplyr::summarise(length(sites))  %>%
pull(sites)
dbs <- df_1 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull()

percent <- round((dbs/((dim(df_c)[1])/(length(unique(df_c$sites)))))*100,digits=3)

cat(
  "\n[Summary]",
  View(df_1))
```

```
df <- cbind(site,dbs,percent)
print(kable(df))

##Boring parts:
suc42<-df_m %>% group_by(sites) %>% distinct(suc42, .keep_all = TRUE) %>% pull(suc42)
sites<-df_m %>% group_by(sites) %>% distinct(suc42, .keep_all = TRUE) %>% pull(sites)
suc42 <- mutate_all(data.frame(suc42), function(x) as.numeric(as.character(x)))
suc42<-cbind(suc42,sites)

##############################################                    OPTIONAL
##############################################

##Optional:
##To get the most accurate input data, set multiplier to be 1 in every site!
##To get more swc values in the final file (at the expense of accuracy), the thresholds for
suc42 should be manually decresead, tethas should be increased!
##Delete all the rows (line 49:59) that you do not need!

##Help to decide how much should we change the thresholds!
##See the distribution of the swc data!
fig <- ggplot(df_1, aes(x = swc, colour = sites)) + geom_density() + geom_rug() +
  ggtitle("Soil water content density below the wilting point")
#+facet_grid(. ~sites, scales = "free_x")
ggplotly(fig)
#gg + geom_histogram(aes(y = ..density.., fill = sites), alpha = 0.2, bins = 50)
#ggplotly()

###########CHANGE VALUES:####################
new_params<-c(F15=0.22,
          F40=0.15,
          F70=0.15,
          G15=0.21,
          G40=0.21,
          A15=0.14,
          A40=0.21,
          A70=0.22,
          V15=0.3,
          V40=0.3,
          V70=0.3)

#multiplier <- c(1)
multiplier <- c(F15=0.8,
           F40=1,
           F70=1,
           G15=1,
           G40=1,
           A15=0.6,
           A40=1,
           A70=1,
           V15=1,
           V40=0.5,
           V70=1)
```

```
###########OPTIMALIZATION DEVICE:###########
##Set 'optimalization' switch to TRUE if you want to change the threshold by percentage
##Set 'optimalization' switch to FALSE if you want to change the threshold's value
manually
optimalization <- T

if(optimalization){
 ##Params has to be set back to its original setting every time before we run it with a
modification!
 params <- suc42
 params <- params %>% mutate(params[1] * multiplier[as.character(sites)])
 print(params)
 }else{
   params <- suc42
   params[1] <- as.numeric(new_params)
   print(params)

 }
################################################
##Creating new data frame by merging it with the newly defined thresholds!
df_13<-df_c
df_13<-df_13[-6]
df_13<-merge(df_13, params, by=c("sites"), all=F, sort=F)
df_13<-df_13[,c(1,2,3,4,5,7,6)]

##Defining blacklisted data!
df_1 <- subset(df_13,!(swc>suc42))

##Saving subset containing incorrect data!
##Please, delete those rows which you might want to keep! If you want to delete all,
please ignore it!
write.xlsx(df_1,
        paste0(output_path,"df_1.xlsx"),col.names=TRUE,row.names=FALSE)
##Reloading subset containing incorrect data!
df_1 <- read.xlsx(paste0(output_path,"df_1.xlsx"),as.data.frame = T,
             sheetIndex = 1)

##Data Overview!
site <- df_1 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>%
pull(sites)
dbs <- df_1 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull()

percent <- round((dbs/((dim(df_c)[1])/(length(unique(df_c$sites)))))*100,digits=3)

cat(
 "\n[Summary]",
 View(df_1))

df <- cbind(site,dbs,percent)
print(kable(df))
```

```
################################################

##Unified, "cleaned" data!
df_1<-df_1 %>%
  replace_with_na_at(.vars=colnames(df_1)[!colnames(df_1) %in% c("date","sites")],
                condition=~.x>-99999999)
df_c<-df_c %>% rows_update(df_1, by =c("date","sites"))

##Soil Water Content graphical representation!
cat(
  "\n====[Cleaning
data]====================================================================
================\n",
  "2.Soil Water Content graphical check!"
)
#dev.new(noRStudioGD = TRUE)
fig1 <- plot_ly(df_c, x=~date, y=~swc, color=~sites,
              type = 'scatter', mode = 'lines',
              connectgaps = FALSE) %>%
  layout(showlegend = TRUE)
fig2 <- plot_ly(df_m, x=~date, y=~swc, color=~sites,
              type = 'scatter', mode = 'lines',
              connectgaps = FALSE) %>%
  layout(showlegend = TRUE)
fig <- subplot(fig1, fig2, shareX = TRUE, nrows = 2, margin = 0.07) %>%
  layout(title = paste0("Soil Water Content graphical comparison"))
ggplotly(fig)

##Cleaning environment a bit!
rm(df_3,df_1,df_13,params,multiplier,new_params,suc42)
##Remove temporary saved files that we do not need!
unlink(paste0(output_path,"df_1.xlsx"))

cat(
  "\nII. phase of cleaning data is finished! Water content below the wilting point is
removed!",
  "\nResults are in df_c file! The blacklisted values are in df_1!",

  View(df_c)
)



####################################################################

##SWCs should be removed for periods it exceeds saturated water content!
cat(
  "\n====[Manually                                                    cleaning
data]====================================================================
=========",
  "\n5.Soil water content should be removed if it exceeds saturated water content!\n"
```

```
)
##Blacklisted data!
df_2 <- subset(df_c,!(swc<tethas))

##Data Overview!
site <- df_2 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>%
pull(sites)
dbs <- df_2 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull()

percent <- round((dbs/((dim(df_c)[1])/(length(unique(df_c$sites)))))*100,digits=3)

cat(
  "\n[Summary]",
  View(df_2))

df <- cbind(site,dbs,percent)
print(kable(df))

##Boring parts:
tethas<-df_m %>% group_by(sites) %>% distinct(tethas, .keep_all = TRUE) %>%
pull(tethas)
sites<-df_m %>% group_by(sites) %>% distinct(tethas, .keep_all = TRUE) %>% pull(sites)
tethas<-mutate_all(data.frame(tethas), function(x) as.numeric(as.character(x)))
tethas<-cbind(tethas,sites)

################################################ OPTIONAL
################################################

##Help to decide how much should we change the thresholds!
##See the distribution of the swc data!
fig <- ggplot(df_2, aes(x = swc, colour = sites)) + geom_density() + geom_rug() +
  ggtitle("Soil water content density above the saturation point")
#+facet_grid(. ~sites, scales = "free_x")
ggplotly(fig)
#gg + geom_histogram(aes(y = ..density.., fill = sites), alpha = 0.2, bins = 50)
#ggplotly()

###########CHANGE VALUES:####################
new_params<-c(F15=0.22,
          F40=0.15,
          F70=0.15,
          G15=0.21,
          G40=0.21,
          A15=0.14,
          A40=0.21,
          A70=0.22,
          V15=0.3,
          V40=0.3,
          V70=0.3)

#multiplier <- c(1)
multiplier <- c(F15=0.3,
        F40=0.3,
        F70=0.3,
        G15=0.3,
        G40=0.3,
        A15=0.3,
        A40=0.3,
        A70=0.3,
        V15=0.3,
        V40=0.3,
        V70=0.3)

###########OPTIMALIZATION DEVICE:###########
##Set 'optimalization' switch to TRUE if you want to change the threshold by percentage
##Set 'optimalization' switch to FALSE if you want to change the threshold's value manually
optimalization <- T

if(optimalization){
  ##Params has to be set back to its original setting every time before we run it with a modification!
  params <- tethas
  params <- params %>% mutate(params[1] * multiplier[as.character(sites)])
  print(params)
}else{
  params <- tethas
  params[1] <- as.numeric(new_params)
  print(params)

}
###############################################
###############################################
##Creating new data frame by merging it with the newly defined thresholds!
df_13<-df_c
df_13<-df_13[-7]
df_13<-merge(df_13, params, by=c("sites"), all=F, sort=F)

##Defining blacklisted data!
df_2 <- subset(df_13,!(swc<tethas))

##Saving subset containing incorrect data!
##Please, delete those rows which you might want to keep! If you want to delete all, please ignore it!
write.xlsx(df_2,
      paste0(output_path,"df_2.xlsx"),col.names=TRUE,row.names=FALSE)
##Reloading subset containing incorrect data!
df_2 <- read.xlsx(paste0(output_path,"df_2.xlsx"),as.data.frame = T,
         sheetIndex = 1)

##Data Overview!
site <- df_2 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull(sites)
dbs <- df_2 %>% dplyr::group_by(sites) %>% dplyr::summarise(length(sites)) %>% pull()
```

```
percent <- round((dbs/((dim(df_c)[1])/(length(unique(df_c$sites)))))*100,digits=3)

cat(
  "\n[Summary]",
  View(df_2))

df <- cbind(site,dbs,percent)
print(kable(df))

################################################
################################################

##Unified, "cleaned" data!
df_2<-df_2 %>%
  replace_with_na_at(.vars=colnames(df_2)[!colnames(df_2) %in% c("date","sites")],
                    condition=~.x>-99999999)
df_c<-df_c %>% rows_update(df_2, by =c("date","sites"))

##Soil Water Content graphical representation!
cat(
  "\n====[Cleaning
data]==============================================================
================\n",
  "2.Soil Water Content graphical check!"
)
#dev.new(noRStudioGD = TRUE)
fig1 <- plot_ly(df_c, x=~date, y=~swc, color=~sites,
               type = 'scatter', mode = 'lines',
               connectgaps = FALSE) %>%
  layout(showlegend = TRUE)
fig2 <- plot_ly(df_m, x=~date, y=~swc, color=~sites,
               type = 'scatter', mode = 'lines',
               connectgaps = FALSE) %>%
  layout(showlegend = TRUE)
fig <- subplot(fig1, fig2, shareX = TRUE, nrows = 2, margin = 0.07) %>%
  layout(title = paste0("Soil Water Content graphical comparison"))
ggplotly(fig)

##Cleaning environment a bit!
rm(df_2,df_13,params,multiplier,tethas)
##Remove temporary saved files that we do not need!
unlink(paste0(output_path,"df_2.xlsx"))

cat(
  "\nIII. phase of cleaning data is finished! Water content above saturation is removed!",
  "\nResults are in df_c file! The blacklisted values are in df_2!",

  View(df_c)
)
```

```
###########################################################################
#######################################

##Saving final cleaned file into a file structure compatible with calibration script we can
use !

#Getting rid of the elements we do not need!
df_c <- df_c[1:3]
#Changing date to the desired look!
date <- format(as.Date(df_c$date, format="%Y-%m-%d"),"%Y.%m.%d")
#Transforming it to a wide format!
df_c<-df_c %>% pivot_wider(names_from = sites, values_from = swc)
#Put everything to a list!
df_c <- map(set_names(site),~select(df_c,starts_with(.x)))
#Changing names!
df_c <-lapply(df_c, function(x) setNames(x, c(paste0("smc",substring(names(x), 2,3)))))
#Giving the list elements a date column!
df_c <- lapply(df_c,`[<-`, 'date', value=date[1:(length(date)/11)])
#Sorting date column to the first place!
#df_c <- lapply(proba3, FUN = function(X){X[c('date', paste0('smc',unique(id)))]})
#Saving it to multiple files!
sapply(seq(df_c), function(x)
  write.xlsx(as.data.frame(df_c[[x]]), paste0(output_path,"observed_smc", names(df_c)[x],
".xlsx"), row.names=FALSE,append=FALSE
  ))
```

# 3. SWAP soft calibration and visualisation tool

```
# SWAP Calibratiob Lite
# Author: Moritz Shore
# Date 21.6.22


# BEING USED BY OPTAIN CASE STUDIES BUT NOT FINISHED!

# MAKE SURE TO UPDATE PATH

# Libraries ----
{
 library(vroom) # read in data
 library(purrr) # map functions
 library(readxl) # read in excel
 library(dplyr) # data manipulation
 library(fs) # for file system access
 library(plotly) # interactive plotting
 library(stringr) # string manipulation
 library(ggdark) # dark theme for plotting
 library(hydroGOF) # Statistics for Hydro-Models
}

# CUSTOM: Global settings ----
{
 # timerange:
 # Would you like to use the whole SWAP modelled timerange, or rather a custom
 # one? Use TRUE for custom
 runswap = T
 custom_date = F
 start = "2016-12-31"
 end = "2018-10-30"
 location = "SWAP_R_Calibration_Tetves"
 path = "c:/Users/Public/SWAP_Tools/UFZDEMO/"
}

# Functions ----
{
 # fixing the formatting of the nibio OneDrive path, which uses the unsupported letter
?
 nibioFix <- function(string){return(str_replace_all(string = string, pattern =
"Bioøkonomi",replacement = "Bio?konomi"))}

read_vap <- function(path, custom_date = FALSE) {
 # reads the SWAP vap file when passed working directory of SWAP folder
 # and returns it in a dataframe format.
 # Read in the .vap file
 # (fails to correctly read in last column, but its not needed to whatever)
 result.vap <- vroom(
  file = path,
    delim = ",",
    comment = "*",
    col_names = F,
    show_col_types = F,
    trim_ws = T,
    guess_max = 100
  )

  # drop incorrectly parsed column
  result.vap <- result.vap[-15]

  # Fix the column names
  colnames(result.vap) <- result.vap[2, ]

  # Remove the erroneous first two rows
  result.vap <- result.vap[-c(1:2),]

  # remove the col names read in as rows if they exist..
  if (length(which(result.vap$date == "cm")) > 0) {
    result.vap <- result.vap[-which(result.vap$date == "cm"), ]
  }
  if (length(which(result.vap$date == "date")) > 0) {
    result.vap <- result.vap[-which(result.vap$date == "date"), ]
  }


  # remove the weird straggler last value
  result.vap <- result.vap[-which(is.na(result.vap$wcontent)), ]

  # force date format
  result.vap$date = as.Date(result.vap$date)

  # force everything else into numeric
  result.vap[2:14] <- result.vap[2:14] %>% map(as.numeric)

  if (custom_date) {
    result.vap %>% filter(date %in% daterange) %>% return()
  } else{
    return(result.vap)
  }
}


depth_recalc <- function(result.vap, smc_depths, observed_smc) {
  # FUNCTION PURPOSE: Recaclulate the swap output depth to the measured depths!
  # input:
  #        dataframe    : result.vap : the result.vap file from SWAP output
  #        vector       : smc_depths : the depths at which smc was observed
  #         dataframe     : observed_Sm : the file for observed smc (to match precision and
colnames)
  # output:  dataframe    : recalculated modelled values at given observed depths
  # creating empty data frame of the right size
  modelled_smc <- tibble(date = daterange)

  # Force the modelled values to have the same precision of the measured:
  # minus 2, to account for chars "0."
  precision = observed_smc[[2]] %>% as.character() %>% nchar() %>% na.exclude() %>%
max() - 2
  result.vap$wcontent <- round(result.vap$wcontent, precision)

  # recalc each depth
  for (depth in smc_depths) {
    # determine what depths swap modelled at
    depths =  unique(result.vap$depth * -1)

    # find the minimum difference(s) to the current measured depth
    minimum_difference = min(abs(depths - depth))

    # have a list of all the difference to compare to
    differences = abs(depths - depth)

    # find the index of the smallest difference(s)
    closest_depths = (depths[which(differences == minimum_difference)]) *
      -1

    # Filter to only the current daterange
    smc_recalc = result.vap %>% filter(date %in% daterange) %>%
      # filter to only the 1-2 close depths
      filter(depth %in% closest_depths) %>% #
      # group by the date to collapse them
      group_by(date) %>%
      # and mean them
      summarise(wcontent = mean(wcontent))

    # apply the recalculated wcontent values to the correct date of the empty DF
    modelled_smc <-
      left_join(x = modelled_smc,  y = smc_recalc, by =  "date")
  }
  # apply the same col names from the obs file
  colnames(modelled_smc) <- colnames(observed_smc)

  # return the recalc'd DF
  return(modelled_smc)
}

df_reformat <- function(modelled_smc, observed_smc) {
  # FUNCTION PURPOSE: Combines and formats the modelled and observed values into
  #                   a data frame friendly format, mainly for use in stat. calc.
  # INPUT: modelled_smc: date, smc15,smc40,smc70 (ie), same for observed
  # OUTPUT: left joined the two input df, with correctly labelled columns
  # set the max bounds (look at the start and end of mod. and measured, and combine
  # for the max timeseries
  if (modelled_smc$date[1] < observed_smc$date[1]) {
    c.start = modelled_smc$date[1]
```

```
 } else{
   c.start = observed_smc$date[1]
 }

 if                          (modelled_smc$date[length(modelled_smc$date)]                         <
observed_smc$date[length(observed_smc$date)]) {
   c.end = modelled_smc$date[length(modelled_smc$date)]
 } else{
   c.end = observed_smc$date[length(observed_smc$date)]
 }

 # merge the observed, and modelled to the extended date range (this method remains
rather untested!!)
 smc_tibble <-
   tibble(date = seq.Date(
     from = as.Date(c.start),
     to = as.Date(c.end),
     by = "day"
   ))
 smc_tibble <-
   left_join(x = smc_tibble, y = modelled_smc, by = "date")
 smc_tibble <-
   left_join(x = smc_tibble, y = observed_smc, by = "date")

 # apply the correct column names
 colnames(smc_tibble)[2:(length(smc_depths) + 1)] <-
   paste0("smc_", smc_depths, "_mod")

 colnames(smc_tibble)[(length(smc_depths) + 2):length(colnames(smc_tibble))] <-
   paste0("smc_", smc_depths, "_mea")

 # and return
 return(smc_tibble)
}

tidy_reformat <-function(modelled_smc,observed_smc = tibble(),smc_depths,tense,run)
{
 # FUNCTION PURPOSE: Combines and formats the modelled and observed values into
 #                a tidy format, mainly for use in plotting with ggplot2
 # input: modelled_smc: dataframe: date, smc15,smc40,smc70 (ie) for modelled
 #       observed_smc: dataframe: date, smc15,smc40,smc70 (ie) for observed
 #       smc_depths: vector: measured depths of SMC
 #       tense: string: is it a past run? "past" or the current run? "current"
 #       run: string: the name of the run
 #
 # output: tibble:  cols: date, value, depth, run, tense, type
 # predef
 smc_tidy <- tibble()

 # for every measured depth:
 for (depth in smc_depths) {
   # only add the observed smc if its passed
```

```
    # be better to make this a seperate function?
    # if length is 0, then its NA, so not passed
    if (length(observed_smc) != 0) {
      # the tibble to add measured rows
      addrows_mea = tibble(
        date = observed_smc$date,
        # +1 offset for the date
        value = observed_smc[[which(depth == smc_depths) + 1]],
        depth = depth,
        run = "measured",
        tense = "measured",
        type = "measured"
      )
      # bind to the master dataframe
      smc_tidy <- rbind(smc_tidy, addrows_mea)
    }

    # dataframe row for adding modelled values
    addrows_mod = tibble(
      date = modelled_smc$date,
      value = modelled_smc[[which(depth == smc_depths) +
                          1]],
      depth = depth,
      run =  run,
      tense = tense,
      type = "modelled"
    )
    smc_tidy <- rbind(smc_tidy, addrows_mod)
  }
  return(smc_tidy)
}

get_hydro_stat <- function(dataframe.df, depth) {
  # FUNCTION PURPOSE: Calculates rmse, nse, pbias, rsr for given depth
  # INPUT: dataframe with format: col: date, measured_depth_1, measured_depth_2,
  #       ...measured_depth_n, modelled_depth1, modelled_depth_2,
  #       ...modelled_depth_n,
  # OUTPUT:  1x5 tibble with format: col: depth, rmse, nse, pbias, rsr
  # parse indicies of the measured and modelled columns
  mea_col <-
    which(colnames(dataframe.df) == paste0("smc_", depth, "_mea"))
  mod_col <-
    which(colnames(dataframe.df) == paste0("smc_", depth, "_mod"))

  # calculate the statistics
  smc_rmse    =    tryCatch({round(rmse(dataframe.df[mod_col],    dataframe.df[mea_col],
na.rm = T), 4)},
                    error = function(e) {return(NA)})
  if(is.nan(smc_rmse)){smc_rmse = NA}

  smc_pbias    =    tryCatch({round(pbias(dataframe.df[mod_col],    dataframe.df[mea_col],
na.rm = T), 4)},
                   error = function(e) {return(NA)})

  smc_rsr = tryCatch({round(rsr(dataframe.df[mod_col], dataframe.df[mea_col], na.rm =
T), 4)},
                  error = function(e) {return(NA)})

  smc_nse = tryCatch({round(NSE(dataframe.df[mod_col], dataframe.df[mea_col]), 4)},
                   error = function(e) {return(NA)})



  # return the values
  return(tibble(
    depth = depth,
    rmse = smc_rmse,
    nse = smc_nse,
    pbias = smc_pbias,
    rsr = smc_rsr
  ))
}

plot_smc <- function(df, depth, stat) {
  c_depth = depth
  df %>% filter(depth == c_depth) %>%
    ggplot() +
    geom_line(aes(
      x = date,
      y = value,
      color = run,
      linetype = type
    )) +
    xlab("Date") + ylab("SMC") +
    # maybe include a % change from last run??? [with red and green]
    ggtitle(
      paste0("SMC @", depth, "cm [", toString(smc_depths), "]"),
      paste0(
        "RMSE (->0):     ",
        stat$rmse[which(stat$depth == c_depth)],
        "\nNSE (->1):    ",
        stat$nse[which(stat$depth == c_depth)],
        "\nPBIAS (->0):  ",
        stat$pbias[which(stat$depth == c_depth)],
        "\nRSR: (->0)    ",
        stat$rsr[which(stat$depth == c_depth)]
      )
    ) +
    dark_theme_gray() +
    theme(legend.position = "right") +
    #scale_size_manual(values = c(2,1 ,0.5))+
    scale_linetype_manual(values = c(modelled = "dotted", measured = "solid"))
}
```

```
stat_plot <- function(data, stat, depth = NA) {
  if (!is.na(depth)) {
    c.depth = depth
    filtered <- data %>% filter(depth == c.depth)
  }
  else{
    filtered = data
    c.depth = "all depths"
  }

  filtered %>%
    ggplot(mapping = aes(
      x = reorder(run, get(stat)),
      y = get(stat),
      patern_fill  = run,
      fill = run,
      alpha = tense
    )) +
    geom_col() +
    geom_label(mapping = aes(
      x = run,
      y = get(stat),
      label  = round(get(stat), 3),
    ),
    alpha = 1) +
    ggtitle(paste0("Statistic: ", stat),
            paste0("detpt @ ", c.depth, "cm")) +
    dark_theme_gray() +
    ylab(stat) +
    xlab("Run") +
    theme(legend.position = "none") +
    theme(axis.text.x = element_text(angle = 90)) +
    scale_alpha_manual(values = c("past" = 0.2, "present" = 1))
}
}


# Main  ----
path <- paste0(path, location)
setwd(path)
dir.create("./runs", showWarnings = F)
dir.create("./results", showWarnings = F)
if(runswap){system2('./SWAP/run_swap',wait = T)}
if (custom_date) {daterange <-seq.Date(from = as.Date(start),to = as.Date(end),by = "day")}

# read the last run and return it to file (can be deleted)
vap_path = paste0(path, "/results/result.vap")
result.vap <- read_vap(vap_path, custom_date)
```

```
time = str_replace_all(Sys.time(), pattern = ":", replacement = "-") %>% str_replace_all(., " ",
"_")
run_name = paste0("./runs/run_") %>% paste0(., time) # transform into path
dir.create(run_name, showWarnings = T)
file.copy(from = "./results/.",
          to = run_name,
          recursive = TRUE)

# determine start and end dates if no custom dates are used
if (!custom_date) {
  start = result.vap$date[1]
  end = result.vap$date[length(result.vap$date)]
  daterange <-seq.Date(from = as.Date(start),to = as.Date(end),by = "day")
}

# READ: OBS smc file ----

observed_smc <-
  read_excel(path = paste0(path, "/observed/observed_smc.xlsx"))
observed_smc$date <- as.Date(observed_smc$date)

smc_depths <- colnames(observed_smc)[-1] %>%
  str_remove(pattern = "smc") %>%
  as.numeric()

if(custom_date){observed_smc <- observed_smc %>% filter(date %in%
as.Date(daterange))}

modelled_smc <- depth_recalc(result.vap = result.vap, smc_depths = smc_depths,
observed_smc = observed_smc)


smc_tibble <- df_reformat(modelled_smc = modelled_smc, observed_smc =
observed_smc)

smc_tidy <- tidy_reformat(
    modelled_smc = modelled_smc,
    observed_smc = observed_smc,
    smc_depths = smc_depths,
    run = "Last_Run",
    tense = "current"
  )

full_df <- tibble()

# find the paths of each run
run_paths <- dir_ls(paste0(path, "/runs")) %>%
  # and add the result.vap file to the path
  paste0(., "/result.vap") %>%
  str_replace_all(pattern = "Bioøkonomi", replacement = "Bio?konomi")
```

```
# loop through each run
for (run_path in run_paths) {
  full_df <-
    # read the result.vap file using own function
    read_vap(run_path) %>%
    # recalculate the depths of the swap output to match the measured
    depth_recalc(
      result.vap = .,
      # piped result df
      smc_depths = smc_depths,
      # parsed observed depth vector
      observed_smc = observed_smc # observed dataframe
    ) %>%
    # resulting recalculation is reformatted to a tidy friendly format
    tidy_reformat(
      modelled_smc = .,
      smc_depths = smc_depths,
      tense = "past",
      run = str_remove(string = run_path, pattern = paste0(path, "/runs/")) %>%
        str_remove(string = ., pattern = "/result.vap")
    ) %>%
    # and bound to the full data frame, row wise
    rbind(full_df, .)
}

# add the "present tense" to the full df
full_df <- rbind(full_df, smc_tidy) # error: mesured will be added twice..

# stats of the last run
smc_stat <- get_hydro_stat(dataframe.df = smc_tibble, depth = 15)


# Calculate the statistics of each past run
hydro_stat_df <- tibble()
for (run in run_paths) {
  for (depth in smc_depths) {
    # get the hydrostatistics from...
    addrow  <- get_hydro_stat(# ...a data frame reformatted talbe
      df_reformat(
        modelled_smc =
          # ...which is sourced from a depth recalculated .vap file
          depth_recalc(
            # ...read here
            result.vap =  read_vap(run),
            # the depths at which to recalculate are sourced here
            smc_depths = smc_depths,
            # and the observed file is sourced from here
            observed_smc = observed_smc
          ),
        # compared to the observed file
        observed_smc = observed_smc
        # and at the current depth
      ),
      depth = depth)

    # label the run with the WD removed path
    addrow$run = str_remove(string = run, pattern = paste0(path, "/runs/")) %>%
str_remove("/result.vap")
    # add te tense
    addrow$tense = "past"

    hydro_stat_df <- rbind(hydro_stat_df, addrow)
  }
}

# do the same for the current run (this could be improved with a function..)
for (depth in smc_depths) {
  last_row <- get_hydro_stat(smc_tibble, depth)
  last_row$run = "last_run"
  last_row$tense = "present"
  hydro_stat_df <- rbind(hydro_stat_df, last_row)
}

# interactive plot
plot_smc(df = full_df,depth = 40,stat = get_hydro_stat(smc_tibble, depth)) %>%
  ggplotly()

# Console printout
{
  cat(

"\n====[Summary]==========================================================
=============]\n",
    "Last run statistics:\n",
    "PBIAS: ",
    hydro_stat_df %>% filter(run == "last_run") %>% summarise(across(c(4),
                                                      mean,
                                                      na.rm = TRUE)) %>% pull(),
    "       Delta: XX%XX",
    "       Best result:  XX.XX @ Run X",
    "\n",
    "NSE:   ",
    hydro_stat_df %>% filter(run == "last_run") %>% summarise(across(c(3),
                                                      mean,
                                                      na.rm = TRUE)) %>% pull(),
    "\n",
    "RMSE:  ",
    hydro_stat_df %>% filter(run == "last_run") %>% summarise(across(c(2),
                                                      mean,
                                                      na.rm = TRUE)) %>% pull(),
    "\n",
    "RSR:   ",
    hydro_stat_df %>% filter(run == "last_run") %>% summarise(across(c(5),
                                             mean,
                                             na.rm = TRUE)) %>% pull(),
    "\n"
  )


  test = hydro_stat_df %>% filter(run == "last_run") %>% summarise(across(c(4),
                                               mean,
                                               na.rm = TRUE)) %>%  pull()
  }
View(hydro_stat_df)
# opens the swap file so you can make changes
shell.exec(file = paste0(path, "/SWAP/Swap.swp"))


# observed_drainage <-read_excel(path = paste0(path,
"/observed/observed_drainage.xlsx"))
# observed_drainage$date <- as.Date(observed_drainage$date)

drainage_sum <- function(result.vap, observed_drainage, drain_plot){
  observed_drainage <- observed_drainage[1:13]
  summed <- result.vap %>% group_by(date) %>% summarise(modelled =
sum(drainage)*10) %>% left_join(., observed_drainage, by = "date")
  summed <- summed[-1,]
  library(reshape2)
  cumsummed <- summed # date carry over
  cumsummed[2:14] <- map(summed[2:14], cumsum) # apply cumsum
  cumsummed_melt<-melt(cumsummed, id.vars = "date") # melt for ggplot
  # apply attribute
  cumsummed_melt$type = NA
  cumsummed_melt$type[which(cumsummed_melt$variable!="modelled")] =
"measured"
  cumsummed_melt$type[which(cumsummed_melt$variable=="modelled")] =
"modelled"
  # plot
  thing <- cumsummed_melt %>% ggplot()+geom_line(aes(x = date, y = value, color =
variable, linetype = type))+
    dark_theme_gray()
  # plot
  return(thing)
}



# drainage_sum(result.vap = result.vap, observed_drainage = observed_drainage,
drain_plot = "r1a") %>% ggplotly()
```

# 4. Script for creating meteorological input files for SWAP in sub-daily resolution

```
#-------------------------------------------------------------------------------
# Name:       extract yearly data from Dotnuvele meteo files for SWAT+
# Purpose:     Interpolating missing records (hours)
#            Removes all days without full 24-hours resords
#            Computes vapour pressure (as given in the Oslo xlsx example)
#            Converts W/m2 to kJ/m2 of solar radiation
#            Edit explanatory ouput text in function cells
#
# Author:      Rasa Idzelyte
#
# Created:     2021-08-24
# Copyright:   (c) Rasa 2021
#-------------------------------------------------------------------------------
import pandas
import os
import datetime
import numpy as np

BASE_DIR  = r'C:\Users\Rasa\Desktop\OPTAIN\WPs\WP_4_3'
DOTNUVELE = 'Dotnuvos_gelez_stotis_station_data.xls'
NO_DATA_VALUE = -99
STATION_NAME  = "'Dotnuvele'" # used also for the output file name

DATA_PATH = os.path.join(BASE_DIR,DOTNUVELE)
OUT_DATA  = pandas.DataFrame()

def out_file_hourly(base_path,data_subset,station,year):

    OUT_PATH = os.path.join(base_path,station[1:-1] + '.' + str(year)[1:])
    now = datetime.datetime.now()
    f = open(OUT_PATH, "w")
    f.write("* Source of data    : Akademija staff\n")
    f.write("* File content      : Meteo data; input file for Swap 3.2 (www.swap.alterra.nl)\n")
    f.write("* File generated by : KU MRI\n")
    f.write("* File generated at : " + now.strftime("%Y-%m-%d %H:%M:%S") + "\n")
    f.write("Date,Record,Rad,Temp,Hum,Wind,Rain\n")
    f.close()

    data_subset[['Dates2','Record',('Saulės radiacija  [W/m2]', 'avg'),
              ('Oro temperatūra, 2 m  [°  C]', 'avg'),'vapour_pres',('Vėjo greitis [m/s]', 'avg'),
              ('Krituliai   [mm]', 'sum')]].to_csv(OUT_PATH, mode='a', sep = ',',  header=False,
index=False,float_format='%.2f')

# Reading measurement data
```

```
DATA = pandas.read_excel(DATA_PATH, header = [0, 1])
DATA.columns = DATA.columns.to_flat_index()
DATA['JUTIKLIAI', 'Date/Time'] = pandas.to_datetime(DATA['JUTIKLIAI', 'Date/Time'],
format='%Y-%m-%d %H:%M:%S')
DATA = DATA.rename({('JUTIKLIAI', 'Date/Time'): 'datetime'}, axis=1)
DATA.sort_values(by=['datetime'],inplace=True)

# checking if there are missing hours and interpolating if there are
min_date = DATA['datetime'].min().date().strftime("%Y-%m-%d")
max_date = DATA['datetime'].max().date().strftime("%Y-%m-%d")
if len(pandas.date_range(start=min_date,
end=max_date,freq='H').difference(DATA['datetime'])) > 0:
   DATA.set_index('datetime', inplace=True)
   upsampled = DATA.resample('H')
   DATA = upsampled.interpolate(method='linear')
   DATA.reset_index(inplace=True)
   del upsampled

DATA['Dates'] = DATA['datetime'].dt.date
DATA['Time']  = DATA['datetime'].dt.time
DATA['Dates'] = pandas.to_datetime(DATA['Dates'], format='%Y-%m-%d')
DATA.reset_index(inplace=True)

# Computing vapour pressure
DATA['vapour_pres'] = 6.1*DATA['Santykinė oro drėgmė  [%]', 'avg']*10**((7.45*DATA['Oro
temperatūra, 2 m  [°  C]', 'avg'])/(234+DATA['Oro temperatūra, 2 m  [°  C]', 'avg']))/1000

# Converting date to sring
DATA['Dates1'] = DATA['Dates'].dt.strftime('%d-%b-%Y')
DATA['Dates2'] = DATA['Dates'].dt.strftime('%Y-%m-%d')

# Remove not full days (not havin 24 hours of records
aaa = DATA.groupby(["Dates2"]).count()
bbb = aaa.loc[(aaa['index'] < 24)]
if not bbb.empty:
   for i in bbb.index.unique():
      DATA = DATA.loc[(DATA['Dates2'] != i)]
del aaa, bbb

# Converting time to number
for i in range(0,24,1):
   DATA.loc[(DATA['Time']==datetime.time(i,0,0)), 'Record'] = i+1
DATA['Record'] = DATA['Record'].apply(np.int64)

# Converting W/m2 to kJ/m2
(https://electronics.stackexchange.com/questions/538921/how-to-convert-calculate-
textkj-textm2-to-textw-textm2)
DATA['Saulės radiacija  [W/m2]', 'avg'] = DATA['Saulės radiacija  [W/m2]', 'avg']*3.6
```

```
# some constant columns
DATA['empty'] = ""
DATA['empty99'] = -99
DATA['station'] = STATION_NAME

# writing output files (don't mind the VisibleDeprecationWarning message)
DATA['year'] = DATA['datetime'].dt.year
years = DATA['year'].unique()
for y in years:
    subset = DATA.loc[(DATA['year'] == y)]
    out_file_hourly(BASE_DIR,subset,STATION_NAME,y)
    del subset
```

# 5. Batch File to run SWAP model and postprocessor

```
@echo off

PATH C:\pest_swap

Swap.exe

C:\Users\cucel\AppData\Local\Programs\Python\Python310\Python.exe OF.py

set /p OF=<.\pf_modeloutputOF.txt

echo CURRENT NSE VALUE is %OF%
```

# 6. Script to Read SWAP Output

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 26 14:37:21 2022
@author: cuceloglu
"""
# ==============================================================================
=
# Libraries
# ==============================================================================
=
import numpy as np
import pandas as pd
import os
import datetime
#from sklearn.metrics import r2_score
#import matplotlib.pyplot as plt

directory = os.getcwd()
os.chdir(directory)

# ==============================================================================
=
# Read SWAP Output
# ==============================================================================
=
def readSWAP(path,ext):
    cols  = ["date", "depth", "wcontent",  "phead",  "hconduc", "drainage",  "rootext",
"waterflux",  "temp",  "solute1",  "solute2", "soluteflux",  "top", "bottom", "day", "dcum"]
    df   = pd.read_csv(path + '/result.' + ext, skiprows=12,names=cols,encoding='cp1252')

    df = df[df["date"].str.contains("cm") == False]
    df = df[df["date"].str.contains("date") == False]
    df = df.reset_index(drop=True)

    for (colname,colval) in df.iteritems():
        df[colname] = df[colname].str.strip()

    #converts string to numeric except first column which is date
    df[cols[1:]] = df[cols[1:]].apply(pd.to_numeric, errors='coerce')
    df['date'] =  pd.to_datetime(df['date'])
    return df
```

```
df = readSWAP('C:/pest_swap/','vap')
df_org = readSWAP('C:/pest_swap/original/','vap')

#
==============================================================================
=
# Organize SWAP Outputs
#
==============================================================================
=

#insert dates for filtering (YYYY-MM-DD)
startDate = '2013-10-10'
endDate   = '2014-03-27'
depth1    = -12.5
depth2    = -17.5
depth_avg = (depth1+depth2)/2

def adjOutput(df,startDate,endDate,depth1,depth2):
    dfx = df.loc[(df['date'].dt.strftime('%Y-%m-%d') >= startDate) & (df['date'].dt.strftime('%Y-
%m-%d') < endDate)]
    #filter for depths
    dfx = dfx[['date','depth','wcontent']]
    dfx['type'] = 'simulated'

    dfx_try = dfx.loc[(dfx['depth']==depth1) | (dfx['depth']==depth2)]
    df_final = dfx_try.groupby(['date']).mean().reset_index()
    return df_final

df_sim = adjOutput(df, startDate, endDate, depth1, depth2)
df_org = adjOutput(df_org, startDate, endDate, depth1, depth2)


# observed
observed   = pd.read_csv('C:/pest_swap/measured.txt')
observed['date'] =  pd.to_datetime(observed['date'])
obsmelt = observed.melt(id_vars ='date', var_name='depth', value_name='value')
obsmelt['type'] = 'observed'
obsmelt['depth'] = obsmelt['depth'].apply(pd.to_numeric, errors='coerce')

obsmelt = obsmelt.loc[(obsmelt['date'] >= startDate) & (obsmelt['date'] < endDate)]
obsmelt = obsmelt.loc[obsmelt['depth']==depth_avg]


df_plot = df_sim
df_plot.rename(columns = {'wcontent':'pest'}, inplace = True)
df_plot['org'] = df_org['wcontent']
df_plot['obs'] = obsmelt['value']
```

```
#df_plot.plot(x="date", y=["obs", "org",'pest'])


#
===============================================================================
=
# Objective Functions
#
===============================================================================
=

of_nse   = 1
of_pbias = 1
of_r2    = 0


def nse(simulation_s, evaluation):
    nse_ = 1 - (np.sum((evaluation - simulation_s) ** 2, axis=0, dtype=np.float64) /
                np.sum((evaluation - np.mean(evaluation)) ** 2, dtype=np.float64))
    return nse_

def pbias(simulation_s, evaluation):
    pbias_ = 100 * np.sum(evaluation - simulation_s, axis=0, dtype=np.float64) /
np.sum(evaluation)
    return pbias_

"""
def r2(simulation_s, evaluation):
    r2_ = r2_score(simulation_s, evaluation)
    return r2_
"""

of_dictionary = {}
if of_nse == 1:
    of_dictionary['nse'] = nse(df_plot['pest'].to_numpy(),df_plot['obs'].to_numpy())
if of_pbias == 1:
    of_dictionary['pbias'] = pbias(df_plot['pest'].to_numpy(),df_plot['obs'].to_numpy())
"""
if of_r2 == 1:
    of_dictionary['r2'] = r2(df_plot['pest'].to_numpy(),df_plot['obs'].to_numpy())
"""

f = open("pf_modeloutputOF.txt", "w")

for key, value in of_dictionary.items():
    print(key, '->', value)
    f.write(str(value))
    f.write('\n')
f.close()
print("simulation is done")
```

# 7. Computation of soil physical and hydrological properties

```
##############################################################
# Script for soil properties calculation using the format of SWAT soil table
# Author/developer: Brigitta Szabó, János Mészáros
# Last modification: 22/06/2022
##############################################################

# installing mandatory libraries for euptfv2
install.packages("devtools")
install.packages("glue")
install.packages("Rdpack")
devtools::install_github("tkdweber/euptf2")


# install and import other necessary packages
packages = c("openxlsx", "stringr")
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)

library(euptf2)

# defining working directory automatically to the path where R file placed
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))


# reading soil data from CSV
soil_prop <- read.csv2("soil_prop_in.csv", dec = ".", sep = ";") # adapt
summary(soil_prop)

# basic structure of the previous file to see variables, data structure etc.
str(soil_prop)

# creating a second soil data frame for later modifications and keep the original one
intact
soil_prop_extended <- soil_prop
summary(soil_prop_extended)

# numeric variables to numeric
soil_prop_extended[, c(9:11, 13:ncol(soil_prop_extended))] = lapply(soil_prop_extended[,
c(9:11, 13:ncol(soil_prop_extended))], FUN = function(y){as.numeric(y)})
summary(soil_prop_extended)
```

```
# replace 0 with NA if 0 means no data
soil_prop_extended[soil_prop_extended == 0] <- NA
summary(soil_prop_extended)

# compute effective bulk density
# reference: Wessolek et al. (2009)
for (j in 1:10) {
  for (i in 1:nrow(soil_prop)){
    if (soil_prop[i,((j*12)+5)] > 1){
      soil_prop_extended[i,(ncol(soil_prop)+j)]     <-     soil_prop[i,((j*12)+2)]     +     0.009     *
soil_prop[i,((j*12)+6)]
    } else {
      soil_prop_extended[i,(ncol(soil_prop)+j)]     <-     soil_prop[i,((j*12)+2)]     +     0.005     *
soil_prop[i,((j*12)+6)] + 0.001 * soil_prop[i,((j*12)+7)]
    }
  }
}

summary(soil_prop_extended)

# computed 0 values mean NA -> rewrite those to NA
soil_prop_extended[soil_prop_extended == 0] <- NA
summary(soil_prop_extended)

# renaming effective bulk density columns
dfnames <- names(soil_prop)
for (i in 1:10) {
  dfnames[length(dfnames)+1] <- paste0("SOL_BD", i, "_eff")
}

# rewriting extended DF names
names(soil_prop_extended) <- dfnames


# new DF validation
soil_prop_extended


# compute soil hydraulic properties
# predict van Genuchten parameters
# use PTF07 of Szabó et al. (2021) if SOL_Z, CLAY, SILT, SAND, BD and SOL_CBN1 are
available, else use the PTF recommended in Table 11 of https://doi.org/10.5194/gmd-14-
151-2021

# if 0 means NA, define it
soil_prop[soil_prop == 0] <- NA
soil_prop$DEPTH_M1 <- soil_prop$SOL_Z1/2
soil_prop2     <-     cbind(soil_prop,     (((soil_prop[,c(25,37,49,61,73,85,97,109,121)])-
(soil_prop[,c(13,25,37,49,61,73,85,97,109)]))/2 + soil_prop[,c(13,25,37,49,61,73,85,97,109)])/10)
names(soil_prop2)[154:length(soil_prop2)] <- paste("DEPTH_M", 2:10, sep="")
soil_prop2$rownum <- 1:nrow(soil_prop2)
```

```
for (i in 1:10){
 input <- soil_prop2[, c("rownum", paste0("DEPTH_M",i), paste0("SOL_BD",i),
paste0("SOL_CBN",i), paste0("CLAY",i), paste0("SILT",i),paste0("SAND",i))]

 # input <- Filter(function(x)!all(is.na(x)), input)
 names(input)[1:7] <- c("rownum","DEPTH_M","BD", "OC", "USCLAY", "USSILT", "USSAND")

 pred_VG1 <- euptfFun(ptf = "PTF07", predictor = input, target = "VG", query =
"predictions")
 names(pred_VG1)[2:6] <- c("THS","THR", "ALP", "N", "M")
 pred_VG <- merge(pred_VG1[, c(1:6,8,10:14)], input[,c(1,2)], by="rownum", all.y=TRUE)

 FC <- pred_VG$THR+(pred_VG$THS-pred_VG$THR)*((1+(((pred_VG$N-1)/pred_VG$N)^(1-
2*pred_VG$N)))^((1-pred_VG$N)/pred_VG$N))
 nam <- paste("FC", i, sep = "")
 assign(nam, FC, envir=.GlobalEnv)

 WP <- pred_VG$THR+((pred_VG$THS-
pred_VG$THR)/((1+pred_VG$THR*(15000^pred_VG$N))^(1-(1/pred_VG$N))))
 nam <- paste("WP", i, sep = "")
 assign(nam, WP, envir=.GlobalEnv)

 SOL_AWC <- FC-WP
 nam <- paste("SOL_AWC", i, sep = "")
 assign(nam, SOL_AWC, envir=.GlobalEnv)

 SOL_K <- (4.65 * (10^4) * pred_VG$THS * (pred_VG$ALP^2))*0.41666001
 nam <- paste("SOL_K", i, sep = "")
 assign(nam, SOL_K, envir=.GlobalEnv)

 # compute albedo
 # method of Gascoin et al. (2009) from Table 6 of Abbaspour, K.C., AshrafVaghefi, S.,
Yang, H. & Srinivasan, R. 2019. Global soil, landuse, evapotranspiration, historical and
future weather databases for SWAT Applications. Scientific Data, 6:263.
 SOL_ALB <- 0.15+0.31*exp(-12.7*FC)
 nam <- paste("SOL_ALB", i, sep = "")
 assign(nam, SOL_ALB, envir=.GlobalEnv)

 # compute USLE erodibility K factor
 # method published in Sharpley and Williams (1990) based on Table 5 of Abbaspour,
K.C., AshrafVaghefi, S., Yang, H. & Srinivasan, R. 2019. Global soil, landuse,
evapotranspiration, historical and future weather databases for SWAT Applications.
Scientific Data, 6:263.
 ES <- 0.2+0.3*exp(-0.0256*input$USSAND*(1-(input$USSILT/100)))
 ECT <- (input$USSILT/(input$USCLAY+input$USSILT))^0.3
 EOC <- 1-(0.25*input$OC/(input$OC+exp(3.72-2.95*input$OC)))
 EHS <- 1-(0.7*(1-input$USSAND/100)/((1-input$USSAND/100)+exp(-5.51+22.9*(1-
input$USSAND/100))))
 USLE_K <- ES*ECT*EOC*EHS
 nam <- paste("USLE_K", i, sep = "")
 assign(nam, USLE_K, envir=.GlobalEnv)
}


num <- str_count(ls(), "SOL_AWC")
sum(num)

pred_SOL_par <- data.frame(mget(c((paste0("SOL_AWC", 1:(sum(num)-1))), (paste0("SOL_K", 1:(sum(num)-1))), (paste0("SOL_ALB", 1:(sum(num)-1))), (paste0("USLE_K", 1:(sum(num)-1))))))

SOL_BD <- soil_prop_extended[, c(153:162)]
names(SOL_BD) <- gsub("_eff", "", names(SOL_BD))

patterns <- c("SOL_AWC","SOL_K", "SOL_ALB", "USLE_K")
soil_prop_final1 <- soil_prop_extended[, -grep(paste(patterns, collapse="|"), colnames(soil_prop_extended))]

soil_prop_final <- cbind(soil_prop_final1[, 1:112], SOL_BD, pred_SOL_par)
soil_prop_final

write.xlsx(soil_prop_final, file = "soil_prop_out.xlsx")


pred_FC_WP <- data.frame(mget(c((paste0("FC", 1:(sum(num)-1))), (paste0("WP", 1:(sum(num)-1))))))
write.xlsx(pred_FC_WP, file = "pred_FC_WP.xlsx")
```

# 8. Script to map soil phosphorus content

```
# -------------------------------------------------------------------------
#       Derive soil P content map for SWAT+ modelling
#
#           author of script: Szabó, B., Kassai, P.
#              first version: 10.12.2021.
#              last changes: 17.01.2022.
# -------------------------------------------------------------------------


# install necessary packages ----
packages = c("raster", "mapview", "dplyr", "sf", "rgdal", "fasterize", "tidyverse", "readxl")
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)

suppressPackageStartupMessages(library(mapview))

# if a function of a package does not work please install the latest version of that package

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
dir()

pathxls <- readxl::read_excel(path = file.path(file.choose()))

CS_name <- pathxls[[1,5]]
dir.create(CS_name)
fileout <- paste0( CS_name, "/")

# 1. load input data ----
# 1.1 load shape of the catchment ----
cs <- st_read(paste0(tools::file_path_sans_ext(pathxls[[2,5]]), ".shp"),
geometry_column="geometry",quiet=TRUE)

# 1.2 load relevant LUCAS Topsoil Survey data, based on the SWAT modelling period ----
# LUCAS Topsoil data sets are available from the OPTAIN cloud
lucas_topsoil <- read.csv2(paste0(tools::file_path_sans_ext(pathxls[[3,5]]), ".csv"))

# 1.3 load relevant LUCAS land cover data, based on the SWAT modelling period ----
# LUCAS land use data sets are available from the OPTAIN cloud
lucas_harmo_uf <- read.csv(paste0(tools::file_path_sans_ext(pathxls[[8,5]]), ".csv"))
# criteria to select samples from LUCAS Topsoil database based on NUTS region (level 0
or 1 or 2)
nuts_r1 <- pathxls[[10,5]]
```

```
nuts_r2 <- pathxls[[11,5]]
nuts_r3 <- pathxls[[12,5]]
nuts_r4 <- pathxls[[13,5]]
nuts_r5 <- pathxls[[14,5]]
nuts_r6 <- pathxls[[15,5]]
nuts_r7 <- pathxls[[16,5]]
nuts_r8 <- pathxls[[17,5]]

# 1.4 load agro-climate zones map ----
cz_map <- raster(paste0(tools::file_path_sans_ext(pathxls[[18,5]]), ".tif"))
# criteria to select samples from LUCAS Topsoil database based on climate
climate1 <- pathxls[[19,5]]
climate2 <- pathxls[[20,5]]
climate3 <- pathxls[[21,5]]
climate4 <- pathxls[[22,5]]
climate5 <- pathxls[[23,5]]

# 1.5 load relevant CORINE, based on the SWAT modelling period ----
# not needed, if local land use map is available
# CORINE data sets are available from the OPTAIN cloud
if (!is.na(pathxls[[4,5]])){
corine <- raster(paste0(tools::file_path_sans_ext(pathxls[[4,5]]), ".tif"))
}

# 1.6 load local land use map ----
if (!is.na(pathxls[[5,5]])){
l_lu_map <- raster(paste0(tools::file_path_sans_ext(pathxls[[5,5]]), ".tif"))
}

# 1.7 load table to recode local land use map categories into CORINE level 2 categories -
---
# this table has to be created by the CS leader for each CS
if (!is.na(pathxls[[7,5]])){
recode_l_lu_map_corine                                                        <-
readxl::read_excel(paste0(tools::file_path_sans_ext(pathxls[[7,5]]), ".xls"))
}

# 1.8 load table to recode LUCAS land cover/ land use categories into CORINE level 2
categories ----
# this table is available from the OPTAIN cloud
recode_lucas_lc1_corine   <-   read.csv2(paste0(tools::file_path_sans_ext(pathxls[[9,5]]),
".csv"))

# 1.9 load soil P content data measured in the catchment, IF AVAILABLE ----
if (!is.na(pathxls[[6,5]])){
p_meas     <-     st_read(paste0(tools::file_path_sans_ext(pathxls[[6,5]]),     ".shp"),
geometry_column="geometry",quiet=TRUE)
}

# 2. add LUCAS land use and land cover categories, climate zone codes to LUCAS Topsoil
dataset ----
```

```
lucas <- merge (lucas_topsoil, lucas_harmo_uf[, c("point_id", "lc1", "nuts0", "nuts1",
"nuts2")], by.x = "point_id", by.y = "point_id")
summary(as.factor(lucas$lc1))

lucas_eu <- st_as_sf(lucas, coords = c("x_laea", "y_laea"))
st_crs(lucas_eu) <- 3035
st_crs(lucas_eu)
# mapview(lucas_eu)

# reproject lucas_eu into crs of cz_map
st_crs(cz_map)
mapview(cz_map)
lucas_eu <- st_transform(lucas_eu, st_crs(cz_map))
st_crs(lucas_eu)

# add climate zone codes with meaning of codes
lucas_eu$cz <- raster::extract(cz_map, lucas_eu, df=TRUE, factors=TRUE)

# 3. recode LUCAS land use and land cover categories into CORINE level 2 categories ---
-
lucas_eu_recode <- merge (lucas_eu, recode_lucas_lc1_corine, by="lc1")
summary(as.factor(lucas_eu_recode$corine_2_lc1))

# 4. compute geometric mean by derived CORINE categories (corine_2_lc1) on lucas_eu
----
geom_mean <- function(data) {
  log_data <- log(data)
  gm <- round(exp(mean(log_data[is.finite(log_data)])), 2)
  return(gm)}

g_mean_lucas_eu <- as.data.frame(with(lucas_eu_recode, tapply(X =
lucas_eu_recode$Olsen_P, INDEX = lucas_eu_recode$corine_2_lc1, FUN = geom_mean)))
g_mean_lucas_eu$corine_2_lc1 <- rownames(g_mean_lucas_eu)
names(g_mean_lucas_eu)[1] <- "g_mean"
g_mean_lucas_eu$n <- summary(as.factor(lucas_eu_recode$corine_2_lc1))
g_mean_lucas_eu

g_mean_lucas_eu <- g_mean_lucas_eu %>% add_row(corine_2_lc1 = c("13", "14", "24", "52"))
g_mean_lucas_eu <- g_mean_lucas_eu[order(g_mean_lucas_eu$corine_2_lc1),]
rownames(g_mean_lucas_eu)<- NULL
g_mean_lucas_eu

# CORINE 11 (urban fabric), 12 (industrial, commercial and transport units) categories:
multiply it with 0.49
g_mean_lucas_eu[(g_mean_lucas_eu$corine_2_lc1 == 11), 1] <-
round(g_mean_lucas_eu$g_mean[(g_mean_lucas_eu$corine_2_lc1 == 11)]*0.49, 2)
g_mean_lucas_eu[(g_mean_lucas_eu$corine_2_lc1 == 12), 1] <-
round(g_mean_lucas_eu$g_mean[(g_mean_lucas_eu$corine_2_lc1 == 12)]*0.49, 2)

g_mean_lucas_eu
```

```
# 5. select relevant samples from LUCAS Topsoil dataset for the CS ----
# 5.1 based on NUTS region ----
if (!is.na(nuts_r1)){
  lucas_sel <- dplyr::filter(lucas_eu_recode, (grepl(nuts_r1, nuts2)| grepl(nuts_r2, nuts2)|
grepl(nuts_r3, nuts2)| grepl(nuts_r4, nuts2) | grepl(nuts_r5, nuts2) | grepl(nuts_r6, nuts2)
| grepl(nuts_r7, nuts2) | grepl(nuts_r8, nuts2)))
}
mapview(lucas_sel[1])
summary(lucas_sel)

# 5.2 based on climate zone ----
if (!is.na(climate1)){
  lucas_sel <- subset(lucas_sel, lucas_sel$cz$category == climate1 | lucas_sel$cz$category
== climate2 | lucas_sel$cz$category == climate3 | lucas_sel$cz$category == climate4)
}

mapview(lucas_sel[1])
summary(lucas_sel)
summary(as.factor(lucas_sel$cz$category))
n <- summary(as.factor(lucas_sel$corine_2_lc1))
n

# 6. compute geometric mean of Olsen P by CORINE level 2 categories ----
g_mean_lucas_sel <- as.data.frame(with(lucas_sel, tapply(X = lucas_sel$Olsen_P, INDEX =
lucas_sel$corine_2_lc1, FUN = geom_mean)))
g_mean_lucas_sel$corine_2_lc1 <- rownames(g_mean_lucas_sel)
names(g_mean_lucas_sel)[1] <- "g_mean"
g_mean_lucas_sel$n <- summary(as.factor(lucas_sel$corine_2_lc1))
g_mean_lucas_sel
check_table <- g_mean_lucas_sel


# 6.1. add the geometric mean Olsen P value to those CORINE level 2 categories which
are missing from the LUCAS selection ----
dif <- setdiff(g_mean_lucas_eu$corine_2_lc1, g_mean_lucas_sel$corine_2_lc1)
dif
g_mean_lucas_sel <- g_mean_lucas_sel %>% add_row(corine_2_lc1 = c(dif))
g_mean_lucas_sel <- g_mean_lucas_sel[order(g_mean_lucas_sel$corine_2_lc1),]
rownames(g_mean_lucas_sel)<- NULL
g_mean_lucas_sel

# 6.2 overwrite geometric mean values for underrepresented non-fertilized land use
categories ----
# CORINE category 11, 12
g_mean_lucas_sel[g_mean_lucas_sel$corine_2_lc1          ==          11,          1]          <-
g_mean_lucas_eu[g_mean_lucas_eu$corine_2 == 11, 1]
g_mean_lucas_sel[g_mean_lucas_sel$corine_2_lc1          ==          12,          1]          <-
g_mean_lucas_eu[g_mean_lucas_eu$corine_2 == 12, 1]

# CORINE category 13, 14
lucas_eu_artificial <- subset(lucas_eu_recode, lucas_eu_recode$corine_2_lc1 < 20)
summary(lucas_eu_artificial)
```

```
g_mean_artificial <- geom_mean(lucas_eu_artificial$Olsen_P)
g_mean_artificial
g_mean_lucas_sel[g_mean_lucas_sel$corine_2_lc1 == 13 | g_mean_lucas_sel$corine_2_lc1
== 14, 1] <- g_mean_artificial
g_mean_lucas_sel

# CORINE category 24
lucas_sel_arable <- subset(lucas_sel, lucas_sel$corine_2_lc1 > 20 & lucas_sel$corine_2_lc1
< 40)
summary(lucas_sel_arable)
g_mean_arable <- geom_mean(lucas_sel_arable$Olsen_P)
g_mean_arable
g_mean_lucas_sel[g_mean_lucas_sel$corine_2_lc1 == 24, 1] <- g_mean_arable
g_mean_lucas_sel

# CORINE category 32, 33, 51, 52
lucas_eu_32_33_5 <- subset(lucas_eu_recode, lucas_eu_recode$corine_2_lc1 > 31 &
lucas_eu_recode$corine_2_lc1 < 40 | lucas_eu_recode$corine_2_lc1 > 50)
summary(lucas_eu_32_33_5)
summary(as.factor(lucas_eu_32_33_5$corine_2_lc1))
summary(as.factor(lucas_eu_32_33_5$iso_countr))
g_mean_32_33_5 <- geom_mean(lucas_eu_32_33_5$Olsen_P)
g_mean_32_33_5
g_mean_lucas_sel[((g_mean_lucas_sel$corine_2_lc1 > 31 &
g_mean_lucas_sel$corine_2_lc1 < 40) | (g_mean_lucas_sel$corine_2_lc1 > 50)), 1] <-
g_mean_32_33_5
g_mean_lucas_sel

# CORINE 41 and 42
lucas_eu_4 <- subset(lucas_eu_recode, lucas_eu_recode$corine_2_lc1 > 40 &
lucas_eu_recode$corine_2_lc1 < 50)
summary(lucas_eu_4)
summary(as.factor(lucas_eu_4$corine_2_lc1))
summary(as.factor(lucas_eu_4$iso_countr))
g_mean_4 <- geom_mean(lucas_eu_4$Olsen_P)
g_mean_4
g_mean_lucas_sel[g_mean_lucas_sel$corine_2_lc1 > 40 &
g_mean_lucas_sel$corine_2_lc1 < 50, 1] <- g_mean_4

g_mean_lucas_sel
g_mean_lucas_eu

str(g_mean_lucas_sel)
g_mean_lucas_sel$corine_2_lc1 <- as.numeric(g_mean_lucas_sel$corine_2_lc1)

rewritten_table <- g_mean_lucas_sel



# 7. add mean Olsen P values to the local land use map ----

# a) if local land use map is not available --> CORINE map is used ----
```

```
if (!exists("l_lu_map")) {
  crs(cs)
  st_crs(corine)
  cs_p <- st_transform(cs, st_crs(corine))
  mapview(cs_p)
  corine_cs <- crop(corine, cs_p)
  mapview(corine_cs) + mapview(cs_p)
  hist(corine_cs)

  corine_cs_spdf <- as(corine_cs, "SpatialPixelsDataFrame")
  hist(corine_cs_spdf@data[,1])
  corine_cs_spdf@data$corine_2_local <- recode(corine_cs_spdf@data[,1],`1` = 11, `2` = 11,
`3` = 12, `4` = 12, `5` = 12, `6` = 12, `7` = 13, `8` = 13, `9` = 13, `10` = 14, `11` = 14, `12` = 21,
`13` = 21, `14` = 21, `15` = 22, `16` = 22, `17` = 22, `18` = 23, `19` = 24, `20` = 24, `21` = 24,
`22` = 24, `23` = 31, `24` = 31, `25` = 31, `26` = 32, `27` = 32, `28` = 32, `29` = 32, `30` =
33, `31` = 33, `32` = 33, `33` = 33, `34` = 33, `35` = 41, `36` = 41, `37` = 42, `38` = 42, `39`
= 42, `40` = 51, `41` = 51, `42` = 52, `43` = 52, `44` = 52, `48` = 999)
  hist(corine_cs_spdf@data[,2])
  summary(as.factor(corine_cs_spdf@data[,2]))
  corine_cs_spdf@data$seq <- c(1:nrow(corine_cs_spdf@data))
  c_r_m2 <- merge(corine_cs_spdf@data, g_mean_lucas_sel, by.x="corine_2_local",
by.y="corine_2_lc1", all.x =TRUE)
  c_r_m2 <- c_r_m2[order(c_r_m2$seq),]
  corine_cs_spdf@data$g_mean_Olsen_P <- c_r_m2$g_mean
  mapview(corine_cs_spdf["g_mean_Olsen_P"])

  c_r_Olsen_P_spdf <- raster(corine_cs_spdf["g_mean_Olsen_P"])
  crs(cs_p)
  crs(c_r_Olsen_P_spdf)
  c_r_Olsen_P_spdf_cs <- projectRaster(c_r_Olsen_P_spdf, crs = crs(cs_p), res =
res(c_r_Olsen_P_spdf))
  crs(c_r_Olsen_P_spdf_cs)
  crop <- crop(c_r_Olsen_P_spdf_cs, extent(cs_p))
  Olsen_P_lucas <- mask(crop, cs_p)
  mapview(Olsen_P_lucas)
  summary(Olsen_P_lucas)
  hist(Olsen_P_lucas)

  writeRaster(Olsen_P_lucas, filename=paste0(fileout, "Olsen_P_lucas_corine.tif"),
format="GTiff", overwrite=TRUE)

  # reproject to EPSG:3035 for upload to the cloud and create metadata .xml
  c_Olsen_P_lucas_3035 <- projectRaster(Olsen_P_lucas, crs = "+init=epsg:3035", res =
res(Olsen_P_lucas))
  writeRaster(c_Olsen_P_lucas_3035, filename=paste0(fileout,
"Olsen_P_lucas_corine_3035.tif"), format="GTiff", overwrite=TRUE)
}



# b) if local land use map is available ----
```

```
if (exists("l_lu_map")) {
  # crop local land use map with catchment shape
  mapview(cs)
  crs(cs)
  mapview(l_lu_map)
  st_crs(l_lu_map)
  cs_p <- st_transform(cs, st_crs(l_lu_map))
  mapview(cs_p)
  l_lu_map_cs <- crop(l_lu_map, cs_p)
  mapview(l_lu_map_cs) + mapview(cs_p) # not all pixels are shown by mapview to
decrease memory needed for plotting

  # 7.1 recode local land use categories into CORINE level 2 categories ----
  l_lu_map_cs_spdf <- as(l_lu_map_cs, "SpatialPixelsDataFrame")
  hist(l_lu_map_cs_spdf@data[,1])
  summary(as.factor(l_lu_map_cs_spdf@data[,1]))
  r <- l_lu_map_cs_spdf@data
  r$seq <- c(1:nrow(r))
  r$l_lu_map_code <- r[,1]
  r_m <- merge (r, recode_l_lu_map_corine, by="l_lu_map_code", all.x =TRUE)

  # add geometric mean of Olsen P to the map based on CORINE level 2
  r_m2 <- merge (r_m, g_mean_lucas_sel, by.x="corine_2_local", by.y="corine_2_lc1", all.x
=TRUE)
  r_m2 <- r_m2[order(r_m2$seq),]
  summary(r_m2)
  l_lu_map_cs_spdf@data$g_mean_Olsen_P <- r_m2$g_mean
  l_lu_map_cs_spdf@data$corine_2_local <- r_m2$corine_2_local
  mapview(l_lu_map_cs_spdf["g_mean_Olsen_P"])
  summary(l_lu_map_cs_spdf["g_mean_Olsen_P"])
  hist(l_lu_map_cs_spdf@data$g_mean_Olsen_P)

  # check mapped Olsen P values by CORINE level 2 categories
  result <- round(with(l_lu_map_cs_spdf@data, tapply(X =
l_lu_map_cs_spdf@data$g_mean_Olsen_P, INDEX =
l_lu_map_cs_spdf@data$corine_2_local, FUN = mean)), 2)
  result

  # crop
  r_Olsen_P_spdf <- raster(l_lu_map_cs_spdf["g_mean_Olsen_P"])
  crs(cs)
  crs(r_Olsen_P_spdf)
  r_Olsen_P_spdf_cs <- projectRaster(r_Olsen_P_spdf, crs = crs(cs), res =
res(r_Olsen_P_spdf))
  crs(r_Olsen_P_spdf_cs)
  crop <- crop(r_Olsen_P_spdf_cs, extent(cs))
  Olsen_P_lucas <- mask(crop, cs)
  mapview(Olsen_P_lucas)

  writeRaster(Olsen_P_lucas, filename=paste0(fileout, "Olsen_P_lucas.tif"), format="GTiff",
overwrite=TRUE)
```

```
  # reproject to EPSG:3035 for upload to the cloud and create metadata .xml
  Olsen_P_lucas_3035 <- projectRaster(Olsen_P_lucas, crs = "+init=epsg:3035", res =
res(Olsen_P_lucas))
  writeRaster(Olsen_P_lucas_3035, filename=paste0(fileout, "Olsen_P_lucas_3035.tif"),
format="GTiff", overwrite=TRUE)


}


# 8. add locally measured soil P content data (IF AVAILABLE) ----
if (exists("p_meas")) {

  # reproject p_meas into CRS of derived Olsen_P_lucas map
  mapview(p_meas)
  st_crs(p_meas)
  p_meas_p <- st_transform(p_meas, crs(Olsen_P_lucas))

  st_crs(p_meas_p)
  mapview(p_meas_p)
  summary(p_meas_p$P_1) # ADAPT ---

  # compute AL-P content from AL-P2O5
  p_meas_p$AL_P <- p_meas_p$P_1*0.4365 # ADAPT if needed ----
  summary(p_meas_p)

  # convert AL-P into Olsen P based on:
  # Sárdi, K., Csathó, P., Osztoics, E. 2009. Evaluation of soil phosphorus contents in long-
term experiments from environmental aspects. Proceedings of the 51st Georgikon
Scientific Conference, Keszthely, Hungary, 807-815.
  p_meas_p$Olsen_P <- p_meas_p$AL_P * 0.5722 - 1.0939 # ADAPT based on table of
transfer functions in guideline ----
  geom_mean(p_meas_p$Olsen_P)
  summary(p_meas_p)

  # merge local measured Olsen P with Olsen P content by land use map
  # rasterize the vector object
  r_p_meas <- fasterize(p_meas_p, Olsen_P_lucas, field = "Olsen_P", fun="max")

  mapview(r_p_meas)
  mapview(Olsen_P_lucas) + mapview(r_p_meas)

  p_corine_meas <- merge(r_p_meas, Olsen_P_lucas, overlap = TRUE)
  mapview(p_corine_meas)

  # clip/crop and mask raster with the CS polygon
  crop_meas <- crop(p_corine_meas, extent(cs))
  Olsen_P_lucas_meas <- mask(crop_meas, cs)
  mapview(Olsen_P_lucas_meas)
  writeRaster(Olsen_P_lucas_meas, filename=paste0(fileout, "Olsen_P_lucas_meas.tif"),
format="GTiff", overwrite=TRUE)

  # reproject to EPSG:3035 for upload to the cloud and create metadata .xml
  Olsen_P_lucas_meas_3035 <- projectRaster(Olsen_P_lucas_meas, crs =
"+init=epsg:3035", res = res(Olsen_P_lucas_meas))
  writeRaster(Olsen_P_lucas_meas_3035, filename=paste0(fileout,
"Olsen_P_lucas_meas_3035.tif"), format="GTiff", overwrite=TRUE)

}


# 9. check number of samples used for the computation and created maps ----
suppressPackageStartupMessages(library(mapview))

check_table
rewritten_table
nrow(lucas_sel)
lucas_sel_map <- mapview(lucas_sel[1])
lucas_sel_map

# view final maps
pal <- grDevices::colorRampPalette(RColorBrewer::brewer.pal(5, "YlGnBu"))

# no measured P value added
Olsen_P_lucas_map <- mapview(Olsen_P_lucas, col.regions = pal(100), legend = TRUE,
na.color = "transparent")
Olsen_P_lucas_map

# measured P value added, shown if measured P value was available
if (exists("Olsen_P_lucas_meas")){
  p_max <- cellStats(Olsen_P_lucas_meas, stat='max', na.rm=TRUE, asSample=TRUE)
  Olsen_P_lucas_meas_map <- mapview(Olsen_P_lucas_meas, col.regions = pal(100), at =
seq(0, p_max+5, 5), legend = TRUE, na.color = "transparent")
}

Olsen_P_lucas_meas_map

# Olsen_P_lucas_map_recol <- mapview(Olsen_P_lucas, col.regions = pal(100), at = seq(0,
p_max+5, 5), legend = TRUE, na.color = "transparent")
# Olsen_P_lucas_map_recol
```

# 9. Script to add soil phosphorus content to Hydrologic Response Units

```
# --------------------------------------------------------------------------
#       Derive soil P content by Hydrologic Response Unit
#       for SWAT+ modelling
#
#          author of script: Szabó, B.
#             first version: 27.06.2022.
#             last changes: 22.07.2022.
# --------------------------------------------------------------------------

library(raster)
library(sp)
library(rgdal)
library(mapview)
library(stringr)

# defining working directory automatically to the path where R file placed
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
dir()

# set in and out files --> adapt
# path of the HRU shape file
path_hru                                                                   <-
"C:/Users/Brigi/Desktop/Brigi_kutatas_egyebek/kutatas/OPTAIN/WP3/task3_1/work/nutr
ients_sol/CS3b" # adapt

# name of the HRU shape file
hru <-"CS3b_HRU"

# location of the Olsen P map
soilP                                                                      <-
"C:/Users/Brigi/Desktop/Brigi_kutatas_egyebek/kutatas/OPTAIN/WP3/task3_1/work/nutr
ients_sol/CS3b/Olsen_P_lucas_meas.tif"

# read HRU shape file
sdata <- readOGR(dsn=path_hru, layer=hru)

# read soil P map
r <- raster(soilP)

# extract P values by HRU polygons -> takes time!
r.vals <- extract(r, sdata)

# calculate mean for each polygon
geom_mean <- function(data) {
  log_data <- log(data)
  gm <- round(exp(mean(log_data[is.finite(log_data)])), 2)
  return(gm)}
```

```
r.mean <- lapply(r.vals, FUN=geom_mean)
mean_OlsenP <- unlist(r.mean)

# add mean values to polygon data
sdata@data$mean_OlsenP <- mean_OlsenP
sdata@data$HRU_ID <- as.factor(sdata@data$HRUS)
summary(sdata@data$HRU_ID)
# hist(as.numeric(sdata@data$HRUS), breaks=2107)
spplot(sdata[13])

sel <- sdata[c(1,11,13)]
HRU_P <- sdata@data[, c(1,11,13)]
HRU_P$NAME_soilnut <- paste("soilnut", HRU_P$HRUS, sep = "")
HRU_P$NAME_soilplant <- paste("soilplant", HRU_P$HRUS, sep = "")

# Write results
writeOGR(sdata, getwd(), "HRU_P", driver="ESRI Shapefile", check_exists=TRUE,
        overwrite_layer=TRUE)
write.csv(HRU_P, file="HRU_P.csv", row.names = F)
```

# 10. Google Earth Engine script to build and apply crop classification model

```
///////////////////////////////////////////////////////////////////////////////////////////////////
// Developer: Janos Meszaros - 2022
// Contact: meszaros.janos@atk.hu

///////////////////////////////////////////////////////////////////////////////////////////////////
// USER DEFINED PARAMETERS

// EASY MODE
// Choose the catchment to process by copying its name to the "catAreaName"
variable:
// Schwarzer_Schops, Petite_Glane, Csorsza, Felso_Valicka, Upper_Zglowiaczka, Pesnica,
// Kebele_Kobiljski, La_Wimbe, Dotnuvele, Cherio, Krakstad, Tetves, Cechticky, Dviete,
Savjaan
var catAreaName = 'Savjaan';

// ADVANCED MODE
// You can change here some parameters if you want to tweak with them,
// legend:
// [5,44,30000,4,5]
// 5: integer number - starting week of the year for the crop season
// 44: integer number - ending week of the year for the crop season
// 30000: integer number - local buffer radius (in meter) around CS polygon to define
area where
//  LUCAS points are collected for training data e.g. 30000 define a 30 km buffer
// 4: integer number - length of period in weeks to collect satellite image for and slice
whole period
//  defined with the first two numbers
// 5: integer number - spatial resolution (in meter) of the final cropmap raster
var catchmentUserD = ee.Dictionary({
  Schwarzer_Schops: [5,44,30000,4,5],
  Petite_Glane: [5,44,30000,4,5],
  Csorsza: [5,44,30000,4,5],
  Felso_Valicka: [5,44,30000,4,5],
  Upper_Zglowiaczka: [5,40,30000,4,5],
  Pesnica: [5,44,30000,4,5],
  Kebele_Kobiljski: [5,44,30000,4,5],
  La_Wimbe: [5,44,30000,4,5],
  Dotnuvele: [5,40,30000,4,5],
  Cherio: [5,44,30000,4,5],
  Krakstad: [5,44,30000,4,5],
  Tetves: [5,44,30000,4,5],
  Cechticky: [5,40,30000,4,5],
  Dviete: [5,44,30000,4,5],
  Savjaan: [5,44,30000,4,5]
});
```

```
////////////////////////////////////////////////////////////////////////////////////////
// Please, do not modify code starting from this line. Thank you.
// If you accidentaly modified it indeed and now you experience erratic operation,
// please contact János Mészáros on his email address: meszaros.janos@atk.hu

////////////////////////////////////////////////////////////////////////////////////////
// 0. Global variables and functions
// dictionary with catchments related hard-coded data
var catchmentD = ee.Dictionary({
  Schwarzer_Schops: ['Schwarzer_Schöps',2015,2021,40,48],
  Petite_Glane: ['Schwarzer_Schöps',2017,2021,40,48],
  Csorsza: ['Csorsza',2015,2021,40,48],
  Felso_Valicka: ['Felso_Valicka',2015,2021,40,48],
  Upper_Zglowiaczka: ['Upper_Zglowiaczka',2015,2021,40,48],
  Pesnica: ['Pesnica',2015,2021,40,48],
  Kebele_Kobiljski: ['Kebele_Kobiljski',2015,2021,40,48],
  La_Wimbe: ['La_Wimbe',2017,2021,48,48],
  Dotnuvele: ['Dotnuvele',2015,2021,40,48],
  Cherio: ['Cherio',2017,2021,40,48],
  Krakstad: ['Dotnuvele',2015,2021,40,48],
  Tetves: ['Tetves',2015,2021,40,48],
  Cechticky: ['Cechticky',2015,2021,40,48],
  Dviete: ['Dviete',2015,2021,40,48],
  Savjaan: ['Savjaan',2015,2021,40,48]
});

// random seed
var seed = 123;

// Sentinel-1 image collection
var S1 = ee.ImageCollection('COPERNICUS/S1_GRD')
.filter(ee.Filter.eq('instrumentMode', 'IW'));

// function for Sentinel-1 image pre-processing
var sentinel = function(img) {
  // masking edges from SAR image
  var edge = img.lt(-30.0);
  var maskedImage = img.mask().and(edge.not());
  img = img.updateMask(maskedImage);
  // adding polarization ratio band
  img = img.addBands(img.expression('VH / VV', {
    'VH': img.select('VH'),
    'VV': img.select('VV'),
  }).rename('VH_VV_ratio'));

  var finalImage = img.select(['VV', 'VH', 'VH_VV_ratio']);

  return finalImage;
};

// catchment area filtering
```

```
var catArea = catchments.filter(ee.Filter.stringContains('name', catAreaName));
// defining crop samples area + specified buffer in dictionary
// satellite images will be clipped for this area
var cropArea = catArea.geometry()
.buffer({distance: ee.List(catchmentUserD.get(catAreaName)).get(2)});
// field boundaries
var fieldBounds = fieldboundaries.filter(ee.Filter.stringContains('name', catAreaName));

// map options
Map.setOptions('HYBRID');
Map.centerObject(catArea);

// week step and total period to generate weeks for
var weekStep = ee.List(catchmentUserD.get(catAreaName)).get(3);
var weeks = ee.List.sequence(ee.List(catchmentUserD.get(catAreaName)).get(0),
ee.List(catchmentUserD.get(catAreaName)).get(1), weekStep);
var beforeweeks = ee.List.sequence(ee.List(catchmentD.get(catAreaName)).get(3),
ee.List(catchmentD.get(catAreaName)).get(4), weekStep);


///////////////////////////////////////////////////////////////////////////////////////////
// I. Training data collection
var lucasPeriod = ee.List([2015, 2018]);

// catchment area filtering
var catAreaTrain = catchments.filter(ee.Filter.stringContains('name',
ee.List(catchmentD.get(catAreaName)).get(0)));
// defining crop samples area + specified buffer
// satellite images will be clipped for this area
var cropAreaTrain = catAreaTrain.geometry()
.buffer({distance: ee.List(catchmentUserD.get(catAreaName)).get(2)});

var sampleDataL = lucasPeriod.map(function(y) {
  // selecting LUCAS points for given year
  var obsPts = LUCASlupts.filter(ee.Filter.eq('year', y))
  .filterBounds(cropAreaTrain);
  // creating time-series multiband images
  // first image for period before actual year
  var imgbeforeL = beforeweeks.map(function(w) {
    var beforemonthFromWeek = ee.Number(w).divide(4.3).ceil();
    var beforeweekDate =
ee.Date.fromYMD(ee.Number(y).subtract(1),beforemonthFromWeek,1);

    var S1fTrain = S1.filterDate(beforeweekDate, beforeweekDate.advance(weekStep,
'week'))
    .filterBounds(cropAreaTrain)
    .map(sentinel);

    return S1fTrain.median();
  });
  // second image for actual year
  var imgL = weeks.map(function(w) {
    var monthFromWeek = ee.Number(w).divide(4.3).ceil();
    var weekDate = ee.Date.fromYMD(y,monthFromWeek,1);

    var S1fTrain = S1.filterDate(weekDate, weekDate.advance(weekStep, 'week'))
    .filterBounds(cropAreaTrain)
    .map(sentinel);

    return S1fTrain.median();
  });
  // combine image list into a single multiband image for "before" and actual period
  var empty = ee.Image().select();
  var imgMBbefore = imgbeforeL.iterate(function(img, result) {return
ee.Image(result).addBands(img);}, empty);
  imgMBbefore = ee.Image(imgMBbefore);
  var imgMB = imgL.iterate(function(img, result) {return
ee.Image(result).addBands(img);}, empty);
  imgMB = ee.Image(imgMB);

  imgMB = ee.Image([imgMBbefore, imgMB]);
  imgMB = imgMB.clip(cropAreaTrain);

  // defining predictor variables present in satellite image mosaic
  var bands = imgMB.bandNames();
  // collecting training data
  var samplesy = imgMB.select(bands).sampleRegions({
    collection: obsPts,
    properties: ['lc1'],
    scale: ee.List(catchmentUserD.get(catAreaName)).get(4),
    tileScale: 16
  });

  return samplesy;
});

// merging yearly sample data into one big feature collection
var sampleData = ee.FeatureCollection(sampleDataL.get(0))
  .merge(ee.FeatureCollection(sampleDataL.get(1)));

// printing number of sample points
print('Number of LUCAS observation points: ', sampleData.size());
// prediction variables from the sample data
var predvar = sampleData.first().propertyNames().slice(1);

//////////////////////////////////////////////////////////////////////////////////////////
// II. Building and testing classifier model
// set of random numbers
var randomL = ee.List([13]);
var splitL = ee.List.sequence(0.0, 0.9, 0.1);

var crossVal = randomL.map(function(cvseed) {
  // adding random values to samples
  var sampleDataCV = sampleData.randomColumn('random', cvseed);
  //10-fold cross validation in one iteration
  return splitL.map(function(splitter) {
    var split_min = ee.Number(splitter);
    var split_max = ee.Number(splitter).add(0.1);
    var testData = sampleDataCV.filter(ee.Filter.gte('random',
split_min)).filter(ee.Filter.lt('random', split_max));
    var trainMin = sampleDataCV.filter(ee.Filter.lt('random', split_min));
    var trainMax = sampleDataCV.filter(ee.Filter.gte('random', split_max));
    var trainData = trainMin.merge(trainMax);

    // random forest training
    var rfm = ee.Classifier.smileRandomForest({
      numberOfTrees: 150,
      minLeafPopulation: 1,
      seed: cvseed
    })
    .setOutputMode('CLASSIFICATION')
    .train({
      features: trainData,
      classProperty: 'lc1',
      inputProperties: predvar
    });


    // testing Random Forest model using dataset table
    // listing represented unique landuse IDs
    var emptyL = ee.List([]);
    var cropIDL = ee.List(testData.iterate(function(cID, result) {return
ee.List(result).add(cID.get('lc1'));}, emptyL)).distinct();
    cropIDL = cropIDL.sort();

    // test and confusion matrix
    var testErrorMatrix = testData.classify(rfm)
    .errorMatrix('lc1', 'classification', cropIDL)
    .array();

    // kappa
    var testKappa = testData.classify(rfm)
    .errorMatrix('lc1', 'classification')
    .kappa();

    // total accuracy
    var testAccuracy = testData.classify(rfm)
    .errorMatrix('lc1', 'classification')
    .accuracy();

    // consumer accuracy
    var testConsumerAccuracy = testData.classify(rfm)
    .errorMatrix('lc1', 'classification', cropIDL)
    .consumersAccuracy();
    testConsumerAccuracy = testConsumerAccuracy.toList().flatten();
    // producer accuracy
    var testProducerAccuracy = testData.classify(rfm)
    .errorMatrix('lc1', 'classification', cropIDL)
    .producersAccuracy();
    testProducerAccuracy = testProducerAccuracy.toList().flatten();

    // final features with accuracy metrics as properties
    return ee.Feature(null, {
      'confusionmatrix': testErrorMatrix,
      'consumeraccuracy': testConsumerAccuracy,
      'cropID': cropIDL,
      'kappa': testKappa,
      'produceraccuracy': testProducerAccuracy,
      'totalaccuracy': testAccuracy
    });
  });
}).flatten();

crossVal = ee.FeatureCollection(crossVal);
print('Accuracy metrics: ', crossVal);
print('Mean accuracy of tests: ' , crossVal.aggregate_mean('totalaccuracy'));
print('Mean kappa of tests: ' , crossVal.aggregate_mean('kappa'));

// exporting accuracy metrics
Export.table.toDrive({
  collection: crossVal,
  description: 'accuracymetricsToDriveCSV',
  folder: 'OPTAIN_export',
  fileFormat: 'CSV',
  fileNamePrefix: catAreaName + '_accuracy_metrics'
});

//////////////////////////////////////////////////////////////////////////////////////////////
// III. Classification on defined period
// list with all years between start and end years
var cmtimeFrame = ee.List.sequence(ee.List(catchmentD.get(catAreaName)).get(1),
ee.List(catchmentD.get(catAreaName)).get(2), 1);

var rfm = ee.Classifier.smileRandomForest({
  numberOfTrees: 150,
  minLeafPopulation: 1,
  seed: seed
})
.setOutputMode('CLASSIFICATION')
.train({
  features: sampleData,
  classProperty: 'lc1',
  inputProperties: predvar
});

// variable importance chart of the Random Forest model
```

```
var varimp = ee.Feature(null, ee.Dictionary(rfm.explain().get('importance')));
var varimpchart = ui.Chart.feature.byProperty(varimp)
  .setChartType('ColumnChart')
  .setOptions({
    title: 'RF Variable Importance',
    legend: {position: 'none'},
    hAxis: {title: 'Variables'},
    vAxis: {title: 'Importance'}
  });
print (varimpchart);

// generating cropmaps for defined years
var cropmapL = cmtimeFrame.map(function(y) {
  // creating time-series multiband images
  // first image for period before actual year
  var clfimgbeforeL = beforeweeks.map(function(w) {
    var beforemonthFromWeek = ee.Number(w).divide(4.3).ceil();
    var beforeweekDate =
ee.Date.fromYMD(ee.Number(y).subtract(1),beforemonthFromWeek,1);

    var S1f = S1.filterDate(beforeweekDate, beforeweekDate.advance(weekStep, 'week'))
    .filterBounds(cropAreaTrain)
    .map(sentinel);

    return S1f.median();
  });
  // second image for actual year
  var clfimgL = weeks.map(function(w) {
    var monthFromWeek = ee.Number(w).divide(4.3).ceil();
    var weekDate = ee.Date.fromYMD(y,monthFromWeek,1);

    var S1f = S1.filterDate(weekDate, weekDate.advance(weekStep, 'week'))
    .filterBounds(cropArea)
    .map(sentinel);

    return S1f.median();
  });

  // combine image list into a single multiband image for "before" and actual period
  var empty = ee.Image().select();
  var clfimgMBbefore = clfimgbeforeL.iterate(function(img, result) {return
ee.Image(result).addBands(img);}, empty);
  clfimgMBbefore = ee.Image(clfimgMBbefore);
  var clfimgMB = clfimgL.iterate(function(img, result) {return
ee.Image(result).addBands(img);}, empty);
  clfimgMB = ee.Image(clfimgMB);

  clfimgMB = ee.Image([clfimgMBbefore, clfimgMB]);

  // actual classification with pretrained RF model
  var classified = clfimgMB.classify(rfm);
  // smoothing
  var classifiedVect = classified.reduceRegions({
    collection: fieldBounds,
    reducer: 'mode',
    scale: ee.List(catchmentUserD.get(catAreaName)).get(4),
    tileScale: 16
  });

  var classifiedFilt = classifiedVect.reduceToImage({
    properties: ee.List(['mode']),
    reducer: 'mode'
  });

  return classifiedFilt;
});

// combine cropmap list into a single multiband cropmap image
var empty = ee.Image().select();
var cropmapMB = cropmapL.iterate(function(img, result) {return
ee.Image(result).addBands(img);}, empty);
cropmapMB = ee.Image(cropmapMB);
// converting years number list to string list
var cmtimeFrameStr = cmtimeFrame.map(function(ynum) {
  return ee.String(ee.Number(ynum).int());
});
// renaming bands for years using previous years string list as new bandnames
cropmapMB = cropmapMB.rename(cmtimeFrameStr);



////////////////////////////////////////////////////////////////////////////////////////////////
// IV. Visualizations
// colorization palettes
var classVis = {min: 1, max: 77, palette: ['white', 'green']};

// first layer of cropmap
Map.addLayer(cropmapMB.select(0), classVis, 'Cropmap', true, 1);

// catchment
Map.addLayer(catArea, {}, 'Catchment', false, 0.75);

// fieldboundaries
//Map.addLayer(fieldBounds, {}, 'Field boundaries', true, 1);

////////////////////////////////////////////////////////////////////////////////////////////////
// V. Exporting results
// Cropmap raster stack export
Export.image.toDrive({
  image: cropmapMB,
  description: 'cropmapStackToDriveGeoTiff',
  fileNamePrefix: catAreaName + '_cropmaps' +
ee.List(catchmentD.get(catAreaName)).get(1).getInfo() + '_' +
ee.List(catchmentD.get(catAreaName)).get(2).getInfo(),
  scale: ee.List(catchmentUserD.get(catAreaName)).get(4).getInfo(),
  region: catArea,
  folder: 'OPTAIN_export',
  fileFormat: 'GeoTIFF',
  maxPixels: 10e10,
  crs: 'EPSG:3035',
  skipEmptyTiles: true
});
```

# 11. Script to merge land use and crop map

```
## Script for creating SWAT compatible land use maps with detailed crops

## Author/developper: Janos Meszaros
## Email: meszaros.janos@atk.hu

# Setting working directory to the folder of this R file
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# year range of crop maps, same as in the Google Earth Engine script
yearlist <- c(2015:2021)

# land use -> SWAT codes LUT
swat_lut <- read.csv(file = list.files(path = "swat_lut/",
                                pattern = "*.csv",
                                all.files = FALSE,
                                full.names = TRUE,
                                recursive = FALSE),
                                header = TRUE)


# national land use map import
landuse <- terra::rast(list.files(path = "landuse_map/",
                             pattern="*.tif$",
                             all.files = FALSE,
                             full.names=TRUE,
                             recursive=FALSE))

# multiyear crop map import
crop <- terra::rast(list.files(path = "crop_map/",
                          pattern="*.tif$",
                          all.files = FALSE,
                          full.names=TRUE,
                          recursive=FALSE))


# changing 0 to NA in cropmap
crop[crop == 0] <- NA

# unifing extent and resolution among rasters
landuse <- terra::resample(landuse, crop, method = "near")


# changing landuse cropland categories to NA
# later these NAs will be filled in with the more detailed crop data
# please change the numeric value according to your land use map
landuse[landuse == 2100] <- NA


# generating and exporting yearly land use vector layer with detailed crops
# loop runs for each band (years) of the crop map
for (i in 1:terra::nlyr(crop)) {
 # merging landuse map with the yearly crop map
 # landuse NAs filled with crop IDs
 landuse_swat <- terra::merge(landuse, crop[[i]])

 # polygonize detailed landuse map
 landuse_swat_polygons <- terra::as.polygons(landuse_swat,
                                   dissolve = TRUE)

 # renaming vector layer attribute
 names(landuse_swat_polygons) <- "SWAT_CODE"

 # renaming numeric IDs to SWAT compatible character codes with inner loop
 for (j in 1:nrow(swat_lut)) {
   landuse_swat_polygons$SWAT_CODE[landuse_swat_polygons$SWAT_CODE ==
swat_lut[j,1]] <- swat_lut[j,2]
 }

 terra::writeVector(landuse_swat_polygons,
                filename = paste0("landuse_swat/landuse_map_swat_", yearlist[i], ".shp"),
                filetype = "ESRI Shapefile",
                overwrite = TRUE)
}
```

# 12. Script to derive metadata XML for multiple data files

```
## XML metadata generator for various GIS file types in OPTAIN project
# based on:
https://github.com/eblondel/geometa/blob/master/inst/extdata/examples/metadata.R


# load or install and load required packages
packages = c("sp", "raster", "sf", "readxl", "httr", "rgdal", "geometa")
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# definition of WGS84 projection system for later reprojection of imported maps
exproj <- raster::crs("+proj=longlat +datum=WGS84 +no_defs")

# definition of warning/error texts list
ermes <- c(
  " ERROR: vector file with missing data in 38th or 43rd row of Excel table",
  " ERROR: grid file with missing data in 43rd row of Excel table",
  " ERROR: tin file with missing data in 34th, 35th, 36th, 37th or 38th or 43rd row of Excel
table",
  " ERROR: textTable file with missing data in 34th, 35th, 36th, 37th or 38th or 43rd row of
Excel table",
  " OK, metadata generated"
)
# warning/error log file init
erlist <- matrix()

# actual data handling starts here ####
# import Excel file for descriptive data on "open file dialog window"
mdxls <- readxl::read_excel(path = file.path(file.choose()), sheet = "Form_R", n_max=42)

# for loop going through the Excel columns ----
for (i in 7:(length(mdxls))) {
  if(mdxls[[40,i]] == "vector" & (is.na(mdxls[[37,i]]) == TRUE | is.na(mdxls[[42,i]]) == TRUE)) {
    erlist[i-6] <- paste0("Data", (i-6), ":", ermes[1])
    next
  } else if (mdxls[[40,i]] == "grid" & is.na(mdxls[[42,i]]) == TRUE) {
    erlist[i-6] <- paste0("Data", (i-6), ":", ermes[2])
    next
```

```
 } else if (mdxls[[40,i]] == "tin" & (is.na(mdxls[[33,i]]) == TRUE | is.na(mdxls[[34,i]]) == TRUE |
is.na(mdxls[[35,i]]) == TRUE | is.na(mdxls[[36,i]]) == TRUE | is.na(mdxls[[37,i]]) == TRUE |
is.na(mdxls[[42,i]]) == TRUE)) {
   erlist[i-6] <- paste0("Data", (i-6), ":", ermes[3])
   next
 } else if (mdxls[[40,i]] == "textTable" & (is.na(mdxls[[33,i]]) == TRUE | is.na(mdxls[[34,i]]) ==
TRUE | is.na(mdxls[[35,i]]) == TRUE | is.na(mdxls[[36,i]]) == TRUE | is.na(mdxls[[37,i]]) ==
TRUE | is.na(mdxls[[42,i]]) == TRUE)) {
   erlist[i-6] <- paste0("Data", (i-6), ":", ermes[4])
   next
 } else {
   # create new metadata environment
   md <- geometa::ISOMetadata$new()

   # create metadata about metadata ----
   md$setLanguage("eng")                         # Metadata language (C)
   md$setDateStamp(ISOdate(substr(Sys.Date(),1,4),                    substr(Sys.Date(),6,7),
substr(Sys.Date(),9,10))) # Metadata date stamp (format YYYY-MM-DD) (M)

   # create metadata point of contact ----
   rp <- ISOResponsibleParty$new()
   rp$setOrganisationName(as.character(mdxls[[1,i]]))    # Metadata point of contact (M)
   rp$setRole("pointOfContact")
   contact <- ISOContact$new()
   address <- ISOAddress$new()
   address$setEmail(as.character(mdxls[[2,i]]))        # E-mail (M)
   contact$setAddress(address)
   rp$setContactInfo(contact)

   md$addContact(rp)

   # create ReferenceSystem object ----
   rs <- ISOReferenceSystem$new()
   rsId <- ISOReferenceIdentifier$new(code = ISOAnchor$new(
     href = as.character(mdxls[[38,i]]),
     name = paste0("EPSG:", as.character(substr(mdxls[[38,i]],39,43)))
   ))

   rs$setReferenceSystemIdentifier(rsId)
   md$setReferenceSystemInfo(rs)

   rs_t <- ISOReferenceSystem$new()
   rsId_t <- ISOReferenceIdentifier$new(code = as.character(mdxls[[39,i]])) # Temporal
Reference System (M)
   rs_t$setReferenceSystemIdentifier(rsId_t)
   md$addReferenceSystemInfo(rs_t)

   # create DataIdentification object ----
   ident <- ISODataIdentification$new()
   ident$setAbstract(as.character(mdxls[[12,i]]))       # Resource abstract (M)
   ident$setLanguage(as.character(mdxls[[18,i]]))       # Resource language(C)
   ident$setCharacterSet("utf8")                     # Character Encoding  (C)
```

```
  ident$addTopicCategory(as.character(mdxls[[21,i]]))    # Dataset topic category (ISO)
(M)

  # add geographic and temporal extent
  extent <- ISOExtent$new()
  # add geographic extent
  # importing data from file path
  # reprojecting GIS layer into WGS84 projection to be able to calculate geographic
bounding box
  if (mdxls[[40,i]] == "grid") {
    gisl <- raster::stack(file.path(mdxls[42,i]))
    gislgeo <- raster::projectRaster(from = gisl, crs = exproj)
    ggex <- raster::extent(gislgeo)
  } else if (mdxls[[40,i]] == "vector") {
    gisl <- rgdal::readOGR(dsn = file.path(mdxls[42,i]))
    gislgeo <- sp::spTransform(x = gisl, CRSobj = exproj)
    ggex <- raster::extent(gislgeo)
  } else if (mdxls[[40,i]] == "tin" | mdxls[[40,i]] == "textTable") {
    ggex <- c(mdxls[[33,i]], mdxls[[34,i]], mdxls[[35,i]], mdxls[[36,i]])
  }

  bbox <- ISOGeographicBoundingBox$new(minx = ggex[1], miny = ggex[3], maxx =
ggex[2], maxy = ggex[4]) # Geographic bounding box (*) (M)
  extent$setGeographicElement(bbox)
  # add temporal extent
  te <- ISOTemporalExtent$new()
  start <- ISOdate(substr(mdxls[[7,i]],1,4), substr(mdxls[[7,i]],6,7), substr(mdxls[[7,i]],9,10)) #
Temporal extent (date from, format: YYYY-MM-DD)(M)
  end <- ISOdate(substr(mdxls[[8,i]],1,4), substr(mdxls[[8,i]],6,7), substr(mdxls[[8,i]],9,10)) #
Temporal extent (date from, format: YYYY-MM-DD)(M)
  tp <- GMLTimePeriod$new(beginPosition = start, endPosition = end)
  te$setTimePeriod(tp)
  extent$setTemporalElement(te)

  ident$setExtent(extent)

  # adding responsible organisation ----
  rp <- ISOResponsibleParty$new()
  rp$setOrganisationName(as.character(mdxls[[15,i]]))    # Responsible organisation (M)
  rp$setRole(as.character(mdxls[[17,i]]))              # Role (M)
  contact <- ISOContact$new()
  address <- ISOAddress$new()
  address$setEmail(as.character(mdxls[[16,i]]))         # E-mail (M)
  contact$setAddress(address)
  rp$setContactInfo(contact)
  ident$addPointOfContact(rp)

  # citation
  ct <- ISOCitation$new()
  ct$setTitle(as.character(mdxls[[11,i]]))             # Resource Title(M)?
  d <- ISODate$new()
  d$setDate(ISOdate(substr(mdxls[[9,i]],1,4),                    substr(mdxls[[9,i]],6,7),
substr(mdxls[[9,i]],9,10)))   # Date of publication/last revision/creation (format YYYY-MM-
DD)(M)
  d$setDateType(as.character(mdxls[[10,i]]))          # Date Type(M)
  ct$addDate(d)
  ct$addIdentifier(ISOMetaIdentifier$new(code = ISOAnchor$new(
    href = as.character(mdxls[[6,i]]),
    name = as.character(mdxls[[6,i]])
  )))
  ident$setCitation(ct)

  # adding access legal constraints
  #for INSPIRE controlled terms on access legal constraints, please browse the INSPIRE
registry:
  # http://inspire.ec.europa.eu/metadata-codelist/LimitationsOnPublicAccess/
  lc <- ISOLegalConstraints$new()              # Conditions applying to access and use (M)
  lc$addUseConstraint("otherRestrictions")
  lc$addOtherConstraint(as.character(mdxls[[27,i]]))
  ident$addResourceConstraints(lc)

  lc2 <- ISOLegalConstraints$new()              # Limitations on public access (M)
  lc2$addAccessConstraint("otherRestrictions")
  lc2$addOtherConstraint(ISOAnchor$new(
    href              =              paste0("http://inspire.ec.europa.eu/metadata-
codelist/LimitationsOnPublicAccess/", as.character(mdxls[[25,i]])),
    name = as.character(mdxls[[26,i]])
  ))

  ident$addResourceConstraints(lc2)

  # add INSPIRE keywords (GEMET Spatial Data Theme)
  inspire_kwd <- ISOKeywords$new()
  inspire_kwd$addKeyword(mdxls[[22,i]])
  th <- ISOCitation$new()
  th$setTitle(
    ISOAnchor$new(
      name = "GEMET - INSPIRE themes, version 1.0",
      href="https://www.eionet.europa.eu/gemet/inspire_themes"
    )
  )
  inspire_date <- ISODate$new()
  inspire_date$setDate(as.Date("2008-06-01"))
  inspire_date$setDateType("publication")
  th$addDate(inspire_date)
  inspire_kwd$setThesaurusName(th)

  ident$addKeywords(inspire_kwd)

  # add GEMET-Concepts keywords
  gemet_kwds <- ISOKeywords$new()
  gemet_kwds$addKeyword(mdxls[[23,i]])        # Keyword GEMET-Concepts (find at:
https://www.eionet.europa.eu/gemet/en/alphabetic/) (M)
```

```
  th <- ISOCitation$new()
  th$setTitle(
    ISOAnchor$new(
      name = "GEMET - Concepts, version 4.1.4",
      href="https://www.eionet.europa.eu/gemet/en/alphabetic/"
    )
  )
  gemet_date <- ISODate$new()
  gemet_date$setDate(as.Date("2020-02-13"))
  gemet_date$setDateType("publication")
  th$addDate(gemet_date)
  gemet_kwds$setThesaurusName(th)

  ident$addKeywords(gemet_kwds)

  # add free keywords
  kwds <- ISOKeywords$new()
  kwds$addKeyword(mdxls[[24,i]])       # Free keywords (comma separated)

  ident$addKeywords(kwds)

  # spatial representation type
  ident$addSpatialRepresentationType(mdxls[[40,i]])  # Spatial representation type (M)

  # add spatial resolution
  res <- ISOResolution$new()                    # Spatial resolution (C)

  if (mdxls[[40,i]] == "grid") {
    resv <- res(gisl)
    if (resv[1] < 1) {
      resu <- "degree"
    } else {
      resu <- "m"
    }
    res$setDistance(ISODistance$new(value = round(resv[1], 1), uom = as.character(resu),
useUomURI = TRUE))
    ident$addSpatialResolution(res)
  } else if (mdxls[[40,i]] == "vector" | mdxls[[40,i]] == "textTable" | mdxls[[40,i]] == "tin") {
    resv <- mdxls[[37,i]]
    res$setEquivalentScale(as.numeric(resv))
    ident$addSpatialResolution(res)
  }

  md$setIdentificationInfo(ident)

  # create dataQuality object ----
  dq <- ISODataQuality$new()
  scope <- ISOScope$new()
  scope$setLevel(as.character(mdxls[[5,i]]))
  dq$setScope(scope)

  #add a report the data quality
```

```
    dc <- ISODomainConsistency$new()
    result <- ISOConformanceResult$new()
    spec <- ISOCitation$new()
    spec$setTitle(as.character(mdxls[[28,i]]))       # Conformity (Specification title) (M)
    d <- ISODate$new()
    d$setDate(as.Date(ISOdate(2010, 12, 8, 1)))      # conformity Date (M)
    d$setDateType("publication")                     # conformity Date type (M)
    spec$addDate(d)
    result$setSpecification(spec)
    result$setExplanation(as.character(mdxls[[29,i]]))  # conformity Explanation (M)
    result$setPass(TRUE)                             # Conformity of the resource [Deegree]  (M)
    dc$addResult(result)
    dq$addReport(dc)

    #add lineage (more example of lineages in ISOLineage documentation)
    lineage <- ISOLineage$new()
    lineage$setStatement(as.character(mdxls[[20,i]]))   # Lineage (M)

    dq$setLineage(lineage)

    md$setDataQualityInfo(dq)

    # add distribution and its format
    distrib <- ISODistribution$new()             # Distribution format [Encoding] (M)
    dto <- ISODigitalTransferOptions$new()
    or <- ISOOnlineResource$new()                # On-line resource [Resource Locator]
    or$setLinkage(as.character(mdxls[[13,i]]))
    or$setOnLineFunction(as.character(mdxls[[14,i]]))
    dto$addOnlineResource(or)
    distrib$setDigitalTransferOptions(dto)

    db <- ISOFormat$new()
    db$setName(ISOAnchor$new(
      name = as.character(mdxls[[41,i]]),
      href=paste0("https://inspire.ec.europa.eu/media-types/", as.character(mdxls[[41,i]]))))
    db$setVersion("unknown")
    distrib$addFormat(db)

    md$setDistributionInfo(distrib)

    # encoding metadata to xml ----
    md$encode()

    # exporting xml to hard drive ----
    md$save(file = paste0(tools::file_path_sans_ext(mdxls[[42,i]]), ".xml"))

    # writing successful exporting into error/warning logfile
    erlist[i-6] <- paste0("Data", (i-6), ":", ermes[5])
  }
}

# writing error/warning messages into log file on hard drive
```

```
write.table(erlist, file = paste0(getwd(), "/error_messages.txt"), append = FALSE, eol = "\n",
row.names = FALSE, col.names = FALSE)
```