Food and Agriculture Organization of the United Nations

PHASE 1

Country Guidelines and Technical Specifications for

Global Soil Nutrient and Nutrient Budget Maps

GSNmap

GLOBAL SOIL PARTNERSHIP

Global Soil Nutrient and Nutrient Budgets maps (GSNmap) Phase I Technical Manual

Food and Agriculture Organization of the United Nations
Rome, 2022

Required citation:
FAO. 2022. *Global Soil Organic Carbon Map* – GSOCmap v.1.6. *Technical report.* Rome. https://doi.org/10.4060/cb9015en

ISBN 978-92-5-135899-3

Cover design: ©FAO/Matteo Sala

Contents

Figures

Tables

Food and Agriculture Organization of the United Nations

PHASE 1

Country Guidelines and Technical Specifications for

Global Soil Nutrient and Nutrient Budget Maps

GSNmap

Licence

Abbreviations and acronyms

BD Bulk density

CEC Cation exchange capacity

CRAN Comprehensive R archive network

DSM Digital soil mapping

GEE Google Earth Engine

GSP Global Soil Partnership

INSII International Network of Soil Information Institutions

ITPS Intergovernmental Technical Panel on Soils

ME Mean error

MAE Mean absolute error

MEC Modelling efficiency coefficient

NDVI Normalized difference in vegetation index

QA/QC Quality assurance/quality check

RMSE Root mean squared error

SOC Soil organic carbon

SOM Soil organic matter

Contributors and reviewers

International Network of Soil Information Institutions

GSNmap working group

Fourth Intergovernmental Technical Panel on Soils

Chapter 1

Presentation

1.1 Background and Objectives

Soil nutrient availability can affect ecosystem carbon cycling, plant phenology, plant diversity and community composition, plant-herbivore and plant-soil-microbe interactions, as well as the structure of trophic food webs (Van Sundert *et al.*, 2020). Thus, the broad range of effects of nutrient availability also affects ecosystem functioning in face of global changes, for instance the response of plants to elevated levels of CO_2 (Vicca *et al.*, 2018).

In the context of agriculture, nutrient availability modulates crop productivity and thus food production. However, the COVID-19 pandemic, current conflicts and devastating extreme weather events triggered by climate change jeopardise achieving sustainable development goal (SDG) 2 (Zero Hunger) by 2030. To date, a total number of around 2.3 billion people are affected by moderate and severe food insecurity (FAO and WHO, 2022). Despite soil nutrient status and availability being vital to the provisioning of ecosystem services, globally-accessible and harmonised datasets on soil nutrient stocks and soil properties that govern nutrient availability are missing.

Therefore, the current global situation requires an increase of food production while preserving natural (soil) resources, lowering greenhouse gas emissions and optimising the use of goods such as fertilisers on agricultural sites (Eisenstein,

2020). Fertiliser prices more than doubled within one year and grain prices increased by around 25 percent (Jan. 2021 - Jan. 2022) (Hebebrand and Laborde, 2022). With the start of the armed conflict in Ukraine in February 2022, this trend became more pronounced. Growing food insecurity and rapidly increasing fertiliser prices underscore the urgent need for informed decision-making and optimised soil nutrient management. However, a large data gap exists in regards to soil nutrient stocks and soil properties that govern nutrient availability. Therefore, FAO's Global Soil Partnership (GSP) has launched the Global Soil Nutrient and Nutrient Budget map (GSNmap) initiative in an endeavour to provide harmonised and finely resolved soil nutrient data and information to stakeholders following a country-driven approach. Up-to-date soil data on the status and spatial trends of soil nutrients and related soil attributes is key to guide policy-making to close yield gaps, and protect local natural resources. Therefore, locally-specific optimisation of soil nutrient and agricultural management are needed (Cunningham *et al.*, 2013). The soil information collected in the GSNmap thereby serves as a cornerstone in delineating priority areas for action and thereby seizes the opportunity to reduce food insecurity, close yield gaps, and reduce environmental costs arising from mismanagement of soil nutrients and especially overfertilisation.

1.2 Global Soil Partnership

The Global Soil Partnership (GSP) was established in December 2012 as a mechanism to develop a strong interactive partnership and to enhance collaboration and generate synergies between all stakeholders to raise awareness and protect the world's soil resources. From land users to policymakers, one of the main objectives of GSP is to improve governance and promote sustainable management of soils. Since its creation, GSP has become an important partnership platform where global soil issues are discussed and addressed by multiple stakeholders at different levels.

The mandate of GSP is to improve governance of the planet's limited soil resources in order to guarantee productive agricultural soils for a food-secure world. In addition, it supports other essential soil ecosystem services in accordance with the sovereign right of each Member State over its natural resources. In order to achieve its mandate, GSP addresses six thematic action areas to be implemented in collaboration with its regional soil partnerships (Figure 1).

The area of work on Soil Information and Data (SID) of the GSP builds an enduring and authoritative global system (GloSIS) to monitor and forecast the condition of the Earth's soil resources and produce map products at the global level. The secretariat is working with the international network of soil data providers (INSII - International Network of Soil Information Institutions) to implement data related activities.

1.3 Country-driven approach and tasks

The GSNmap initiative will be jointly implemented by the International Network of Soil Information Institutions (INSII) and the GSP Secretariat. The process will be country-driven, involving and supporting all Member States in developing their national GSNmap data products. The GSNmap products will be developed following a two phase approach:

- Phase I: development of soil nutrient and associated soil property maps;
- Phase II: quantification, analysis, projections of nutrient budgets for agricultural land use systems at national, regional and global scale.

These guidelines only concern GSNmap Phase I, while the guidelines for the GSNmap Phase II will be published in the fourth quarter of 2022. The Technical Manual covers the generation of soil property maps for the soil attributes specified in table 1.1. It is based on the contribution of each element to the average plant content.

Depending on national data availability and technical capacities, ad-hoc solutions will be developed by the GSNmap WG to support countries during the national GSNmap production and/or harmonisation phase. Where possible, GSP Secretariat will use publicly available data to gap-fill the areas which are not covered by the national submissions unless the country requests to be left blank on the GSNmap products.

1.4 How to use this book

The present document is a technical manual on the phase I of the GSNmap initiative. It provides the scientific background on the importance of soil nutrients and guidance on the digital soil mapping techniques to map nutrients

Table 1.1: Mandatory soil attributes and units of the phase I Global Soil Nutrient and Nutrient Budget Maps

Soil property	property id	Unit
Total N	n_0_30	ppm
Available P	p_0_30	ppm
CEC	cec_0_30	cmol(c)/kg
pH (water)	ph_0_30	/
Clay	clay_0_30	%
Silt	silt_0_30	%
Sand	sand_0_30	%
Soil Organic Carbon	soc_0_30	%
Bulk density	bd_0_30	g/cm3
Available K	k_0_30	ppm

and soil properties that govern nutrient availability. It also comprises a compendium with all necessary scripts to generate national GSNmaps. These scripts are described step-by-step in 4 steps that cover soil data preparation (Step 1), covariate download (Step 2), the mapping process itself (Step 3), and the automatic generation of national reports (Step 4). The general workflow is shown in Figure 1.1.

The chapters are structured as following:

- Chapter 1 provides general information about the GSNmap initiative as another activity of the GSP.
- Chapter 2 focuses on the scientific state-of-the-art in terms of soil nutrients and soil nutrient mapping.
- Chapter 3 and 4 introduce the software requirements and the concept of digital soil mapping.
- Chapter 5 to 7 guide the reader through the nutrient mapping exercise of GSNmap Phase I providing step-by-step instructions.
- Chapter 8 explains how the national GSNmaps are reported to the GSP.
- Annex I serves as a repository for the complete scripts needed for the GSNmap.
- Annex II provides alternative step-by-step instructions for the special case of soil data without point coordinates.

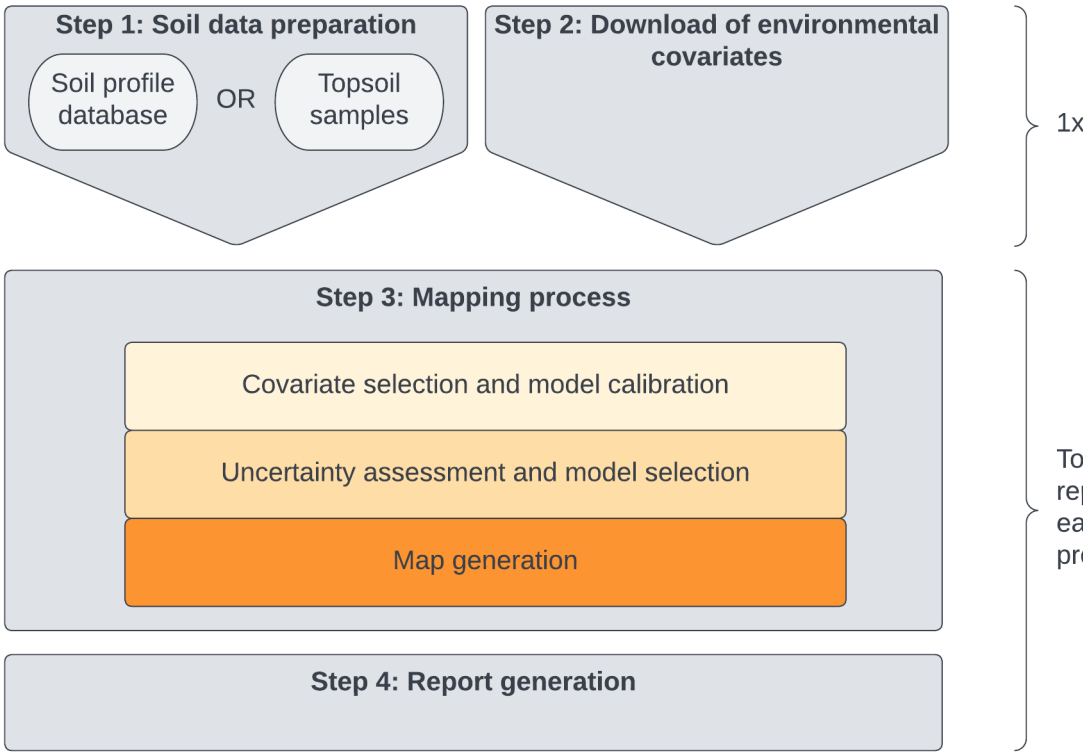


Figure 1.1: Overview of the steps to follow for the GSNmap generation.

The GSNmap Technical Manual is structured as a practical document to be used by national experts in the endeavour to employ digital soil mapping techniques to generate national nutrient maps based on a common methodology. The concept of digital soil mapping presented here can however be also used in mapping exercises that focus on other soil properties and is therefore also relevant to scientists and digital soil mappers. The training material and the folders of the technical manual can be downloaded as .zip file here: https://github.com/FAO-GSP/GSNmap-TM/archive/refs/heads/main.zip. Alternatively, the GitHub repository can be cloned to your local device by using the following link: https://github.com/FAO-GSP/GSNmap-TM.git.

1.5 Training material

The train material of this book is located in the GSNmap-TM GitHub repository. To download the input files and **R** scripts, clone the repository or click on this link, save the ZIP file and extract its content in a folder, preferable close to the root of your system, such as `"C:/GIT/"`.

Chapter 2

Soil Nutrients

2.1 Definition of soil nutrients

In theory, soil nutrients are defined as those chemical elements that are *essential* to plant growth (Arnon and Stout, 1939; Liebig, 1841). First, von Liebig (1841) declared nitrogen (N), sulphur (S), phosphorus (P), potassium (K), calcium (Ca), magnesium (Mg), silicon (Si), sodium (Na) and iron (Fe) as being essential. However, these findings lacked experimental research and were based on merely observational studies. Furthermore, if plant uptake is the only criteria for essentiality, the definition disregards the fact that plants also take up innecessary or even toxic elements. Therefore, stricter criteria such as the one by Arnon and Stout (1939) were defined. They postulated that three criteria need to be met for an *essential mineral nutrients* (Mengel and Kirkby, 2012): 1. Nutrient must be required by plants to complete their life cycle; 2. Nutrient must be irreplaceable; and 3. Nutrient must be involved in the plant metabolism.

Following this defintion to date the following nutrients would be considered essential for higher plants (Mengel *et al.*, 2001): carbon (C), hydrogen (H), oxygen (O), N, P, S, cobalt (Co), K, Ca, Mg, Fe, manganese (Mn), copper (Cu), Si, zinc (Zn), molybdenum (Mo), boron (B), chlorine (Cl), nickel (Ni), Na. However, Co, Si, Ni, and Na are not considered essential for all plants.

Other definitions used biochemical functions for classification purposes (Mengel *et al.*, 2001). Here, four nutrient groups are distinguished: 1. major constituents

Table 2.1: Classification of major and micronutrients by FAO (2022).

Major nutrients	Micronutrients
N	Fe
P	Mn
K	Zn
Ca	Cu
Mg	B
S	Mo
	Cl

of organic material (C, H, O, N, S); 2. nutrients that are involved in esterification of alcohol groups (P, B, Si); 3. nutrients that establish an osmotic potential (ions) (K, Na, Ca, Mg, Mn, Cl); and 4. nutrients that enable electron transport (ions or chelates) (Fe, Cu, Zn, Mo).

Still, the most common classification of soil nutrients is based on the absolute quantities of an element that a plant takes up resulting in macro- and micronutrients (Mengel and Kirkby, 2012). Despite being widely used, the definition has several shortcomings as also toxic elements can be taken up in greater quantities (e.g. Al). Furthermore, the threshold definition between macro- and micronutrients is somewhat arbitrary (Mengel and Kirkby, 2012). It is important to point out that the discussion on how to accurately define *essential* nutrients is ongoing as recent contribution to the topic show (Brown, Zhao and Dobermann, 2022). The generation of the GSNmaps is oriented by the recently published report on the state of the art of soils for nutrition (FAO, 2022) and is shown in Table 2.1. It is based on the contribution of each element to the average plant content.

2.2 Soil properties governing nutrient availability

The uptake of nutrients by plants is regulated in parts by the organism itself as for instance shoot growth is coupled with root growth (Wang *et al.*, 2007). Still, soil properties mediate nutrient mobility and conditions at the plant-soil interface. The most important soil properties that determine nutrient availability are physicochemical properties such as soil pH, cation exchange capacity (CEC),

soil texture, soil organic matter (SOM) content, and bulk density (BD). Most nutrients are taken up in their ionized form (Robertson *et al.*, 1999). Therefore, the chemical characterization of the soil solution is key to understand nutrient dynamics and uptake. Soil pH, as a measure of exchangeable hydrogen protons (H^+), is a crucial parameter to determine the acidity of the soil solution that can inhibit or mediate nutrient uptake. For instance, very low pH values of around 4 decreased the uptake of (basic) cations such as Ca or Mg by paddy rice, wheat, corn, common bean and cowpea whereas lower pH values favoured the uptake of Zn, Fe, and Mn. At higher pH values the uptake of cations was enhanced (Fageria and Knupp, 2014). The CEC, as a measure of exchangeable cations (e.g K^+, Mg^{2+}, Ca^{2+}, etc.) available in the soil solution and attached to soil particles is a complementary parameter of nutrient availability in soils (Robertson *et al.*, 1999). The CEC informs on the capacity of soils to retain positively charged nutrients (basic cations) and thus gives information on how strong a soil can buffer subsequent acidification. This retention and buffer capacity is strongly linked to soil texture. High clay contents usually lead to higher CEC and thus higher cation retention. Conversely, sandy textured soils strongly rely on soil organic matter (SOM) content that has high CEC to retain cations. SOM content further augments aeration of soils due to its low density and provides high specific surface area to retain nutrients. Finally, BD is key to nutrient availability as it governs facilitates or inhibits root growth and thus nutrient uptake by plants. Due to its impact on soil porosity, BD also governs microbial activity (through aeration) and water infiltration that defines nutrient mobility.

Chapter 3

Setting-up the software environment

This chapter provides an overview on the software required to map soil nutrients and associated soil properties. The tools are open source and can be downloaded and installed by users following the steps that are described here. The instructions given are for Microsoft Windows operational systems. Instructions for other operational systems (e.g. Linux Flavours, MacOS) can be found through free online resources.

3.1 Use of R, RStudio and R Packages

R is a language and environment for statistical computing created in 1992. It provides a wide variety of statistical (e.g. linear modeling, statistical tests, time-series, classification, clustering, etc.) and graphical methods, and has been constantly extended by an exceptionally active user community.

3.1.1 Obtaining and installing R

Installation files and instructions can be downloaded from the Comprehensive R Archive Network (CRAN).

1. Go to the following link https://cran.r-project.org/ to download and install **R**.
2. Pick an installation file for your operational system.
3. Choose the "*base*" distribution of R (particularly if it is the first time you install **R**).
4. Download the R installation file and open the file on your device.
5. Follow the installation instructions.

3.1.2 Obtaining and installing RStudio

Beginners will find it very hard to start using **R** because it has no Graphical User Interface (GUI). There are some GUIs which offer some of the functionality of **R**. **RStudio** makes **R** easier to use. It includes a code editor, debugging and visualization tools. Similar steps need to be followed to install **RStudio**.

1. Go to https://www.rstudio.com/products/rstudio/download/ to download and install **RStudio**'s open source edition.
2. On the download page, *RStudio Desktop, Open Source License* option should be selected.
3. Pick an installation file for your platform.
4. Follow the installation instruction on your local device.

The **RStudio** interface is structured by four compartments (see Fig. 3.1). The code editor is located in the upper left. Scripts that contain codes are displayed here. New scripts can be opened by clicking on the left most *New* button in the quick access tool bar (highlighted in green). Lines of code can be executed by clickinig on *Run* (highlighted in blue) or by pressing *ctrl + enter* on your keyboard. The output of scripts or lines of code that are executed is displayed in the window below the code editor: the console (bottom left). This part of the interface corresponds to the **R** software that were installed previously. When working in **R**, it is central to work with so-called objects (for instance vectors, dataframes or matrices). These objects are saved in the global environment that is displayed in the top right panel. Finally, the **R** software offers a broad range of powerful tools for visualisation purposes. Graphs or maps that are generated by scripts/codes, are displayed in the bottom right panel.

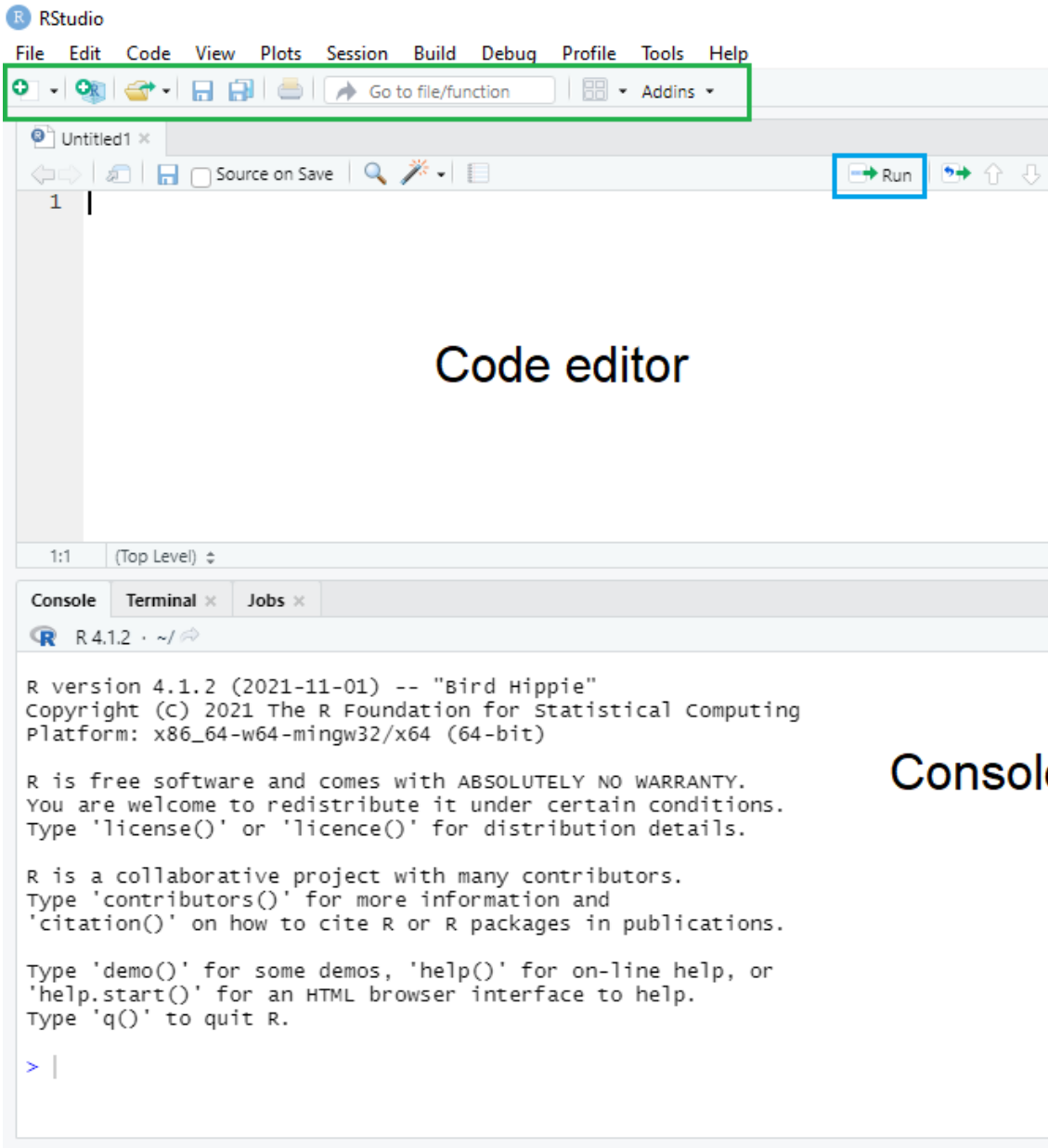


Figure 3.1: R Studio interface.

3.1.3 Getting started with R

- **R** manuals: http://cran.r-project.org/manuals.html
- Contributed documentation: http://cran.r-project.org/other-docs.html
- Quick-**R**: http://www.statmethods.net/index.html
- Stackoverflow **R** community: https://stackoverflow.com/questions/tagged/r

3.2 R packages

When you download **R**, you get the basic **R** system which implements the **R** language. **R** becomes more useful with the large collection of packages that extend the basic functionality of it. **R** packages are developed by the **R** community.

refer to: * *tidyverse* book (R for data science) * *caret* (broad range of statistical learning functions) * *R spatial*: https://rspatial.org/ (R packages for spatial data operations)

The primary source for **R** packages is CRAN's official website, where currently about 12,000 available packages are listed. For spatial applications, various packages are available. You can obtain information about the available packages directly on CRAN with the 'available.packages()' function. The function returns a matrix of details corresponding to packages currently available at one or more repositories. An easier way to browse the list of packages is using the *Task Views* link, which groups together packages related to a given topic.

Packages come along with extensive documentation that is very helpful to understand and solve error messages. To access information on functions or packages, type "?[Package or Function name]" in the console. The information on the package and/or function can then be accessed in the bottom right panel under "Help" (see Fig. 3.1). In addition to that, the *R documentation* website (https://www.rdocumentation.org/) provides more extensive help and gives clear overviews on all functions comprised in a certain package.

3.3 GEE - google earth engine

Google earth engine (GEE) provides a large range of remote sensing datasets for users. It allows to use the GEE code editor to run computations using the

google servers. The high computational power of these servers enables users with limited computational caapacities to run complex calculations. A user account must be created to use the code editor. This step can take some time. Once the account is validated, scripts can be written in the code editor using the Javascript language. An extensive array of instructions and guides are available on the platform. Alternatively, the Python language can be used to interact with the data.

The code editor interface is structured by three panels and a map viewer (see Fig. **??**). The left panel is structured in "Scripts", "Docs", and "Assets". Under "Scripts" users can organize and save the scripts they wrote for specific purposes. "Docs" provides further information on so-called "server-side" functions that can be used to manipulate the data. Finally, in "Assets" users can upload own spatial data in common formats such as shapefiles (.shp) or raster files (.tif). The middle panel contains the scripts that can be run by clicking on the "Run" button. The right panel is composed of three functionalities. The "Inspector" provides basic information on a pixel of a layer displayed in the map below. The information consists of longitude, latitude, and - if layers are loaded - values of the pixel. The "Console" is the place where certain commands expressed in the code are shown. The most common expressions shown here are print() commands or figures derived from the loaded data. Finally, the "Tasks" button shows all tasks that were formulated in the code/script and are to be submitted to the server for computation. Once a task is submitted, the user has to click on the "Run" button appearing in the "Tasks" section to submit the task to the server. In addition to that, the data catalog can be accessed via the search bar on the top of the page. Here, key information on the available datasets, origin, resolution and related publications can be found.

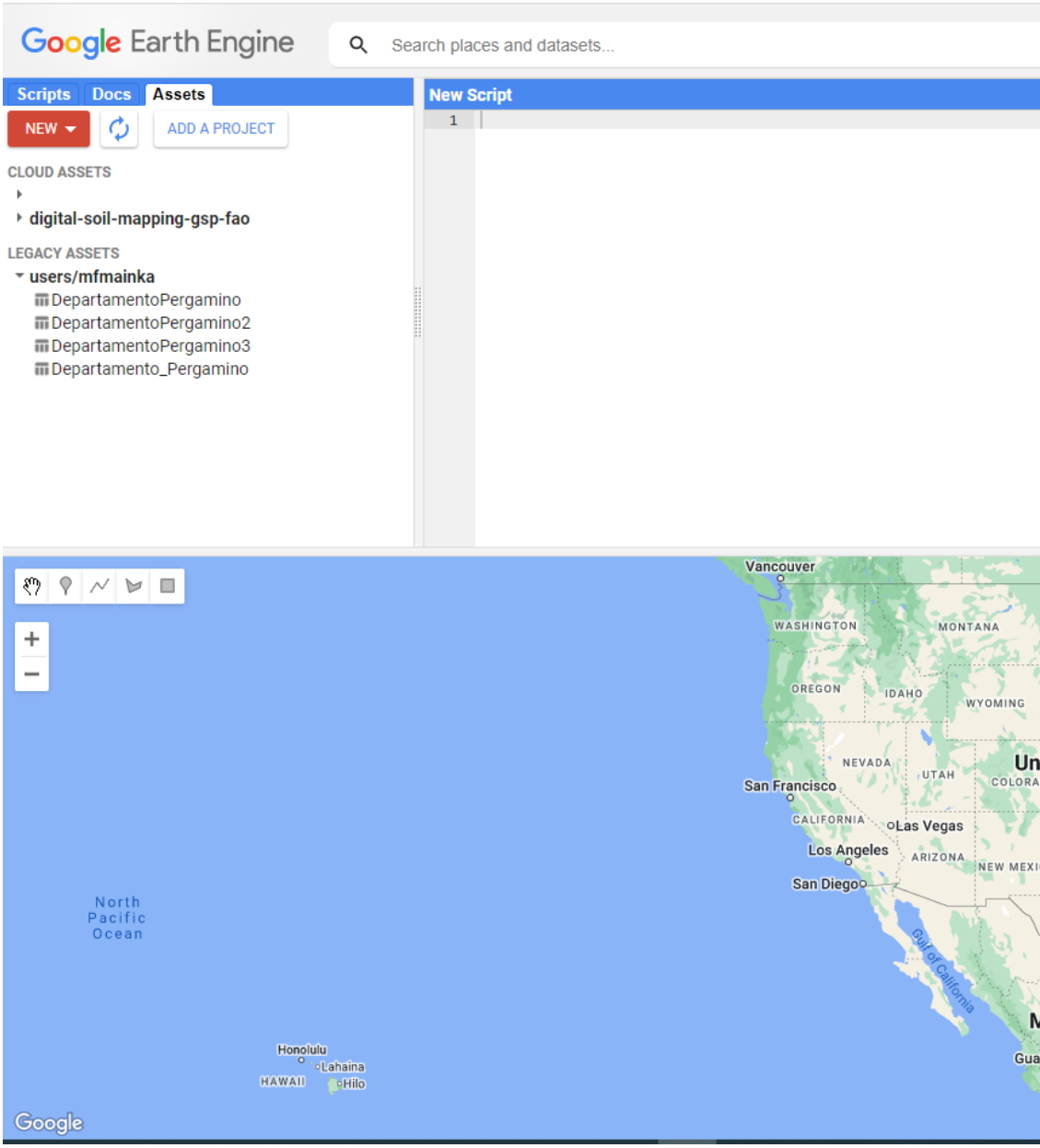


Figure 3.2: Google Earth Engine code editor.

Chapter 4

Digital Soil Mapping

4.1 Principles

Digital soil mapping (DSM) is a methodological framework to create soil attribute maps on the basis of the quantitative relationships between spatial soil databases and environmental covariates. The quantitative relations can be modelled by different statistical approaches, most of them considered machine learning techniques. Environmental covariates are spatially explicit proxies of soil-forming factors that are employed as predictors of the geographical distribution of soil properties. The methodology has evolved from the theories of soil genesis developed by Dokuchaev (1883) in his work the Russian Chernozems, which later were formalised by Jenny (1941) with the equation of the soil-forming factors. The conceptual equation of soil-forming factors has been updated by McBratney, Santos and Minasny (2003) as follow:

$$S = f\left(s, c, o, r, p, a, n\right) \tag{4.1}$$

Where S is the soil classes or attributes (to be modelled) as a function of "s" as other soil properties, "c" as climatic properties, "o" as organisms, including land cover and human activity, "r" as terrain attributes, "p" as parent material, "a" as soil age, and "n" as the geographic position.

4.2 Environmental covariates

There is an constantly increasing range of global datasets that can be used as environmental covariates. Covariates usually provide information on the soil forming factors. However, they are always only an approximation to the reality in the field. The selection of covariates aims to give the most accurate picture of the reality and thus complement each other. In the case of climatic covariates for instance, useful covariates should not only cover the long-term mean annual temperature or precipitation over an climatic reference period (30 years) but also inform about seasonal patterns or even diurnal variability. Still, when selecting covariates one has to keep in mind that there is a trade-off between accurate representation of reality and overfitting the model used for modelling.

4.3 Machine learning techniques

A broad range of modelling approaches coexist in order to establish quantitative relationships between environmental covariates and the target soil properties to be mapped. The plethora of methods cannot be listed here as it was summarised in multiple review papers (Khaledian and Miller, 2020; Lamichhane, Kumar and Wilson, 2019; Ma *et al.*, 2019; Padarian, Minasny and McBratney, 2019; Wadoux, Minasny and McBratney, 2020). Traditionally, multiple linear regression models can be used to quantify the relationships which continues to be the most applied mapping method to map for instance soil organic carbon (Lamichhane, Kumar and Wilson, 2019). In addition to that, regression Kriging methods combine linear regressions and an stochastic interpolation of the regression residuals based on their spatial autocorrelation (Yigini *et al.*, 2018). However, machine learning algorithms with more flexible assumptions, i.e. non-linear relationships, have become more and more popular as the mapping performance was substantially improved and the versatility of the algorithms can be detect more complex relationships. Among the most commonly used non-linear machine learning models is random forest (Breiman, 2001). The random forest algorithm splits a dataset into subsets and uses a random selection of covariates (predictors) to identify homogeneous groups. The procedure of classifying is repeated many times and in the end the prediction is averaged. Finally, quantile regression forests (QRF) derive from random forest models (Meinshausen, 2006). The benefit of QRF is the ability to predict not only the mean of the prediction but also to provide more information on the uncertainty and probability

distribution.

4.4 Mapping of soil nutrients and associated soil attributes

DSM has been used to produce maps of soil nutrients at regional to continental scales. For instance, Hengl *et al.* (2017) predicted 15 soil nutrients at a 250 m resolution in Africa using a random forest model (Wright, Ziegler and König, 2016). The soil nutrient observations were collected for topsoils at locations that were unevenly distributed over the continent and a set of spatially-explicit environmental covariates including soil properties. In 2021, the map resolution was increased to 30 x 30 m by using additional soil samples (Hengl *et al.*, 2021). In Europe maps of chemical soil properties, including macronutrients like potassium and phosphorus, were mapped based on a gaussian process regression using the LUCAS soil database (Ballabio *et al.*, 2019). Global efforts to map nutrients in a harmonised way are suffering of constraints due to limited availability of appropriate soil data. The country-driven approach of the GSP has therefore the potential to improve data availability through the country-driven approach as it uses largely unexplored soil data, a harmonised mapping approach combined with national expertise on the regional soil resources. Therefore, in this technical manual, we present a DSM framework to map soil nutrients and associated properties using soil observations with latitude and longitude coordinates (point-support) (Figure 4.1).

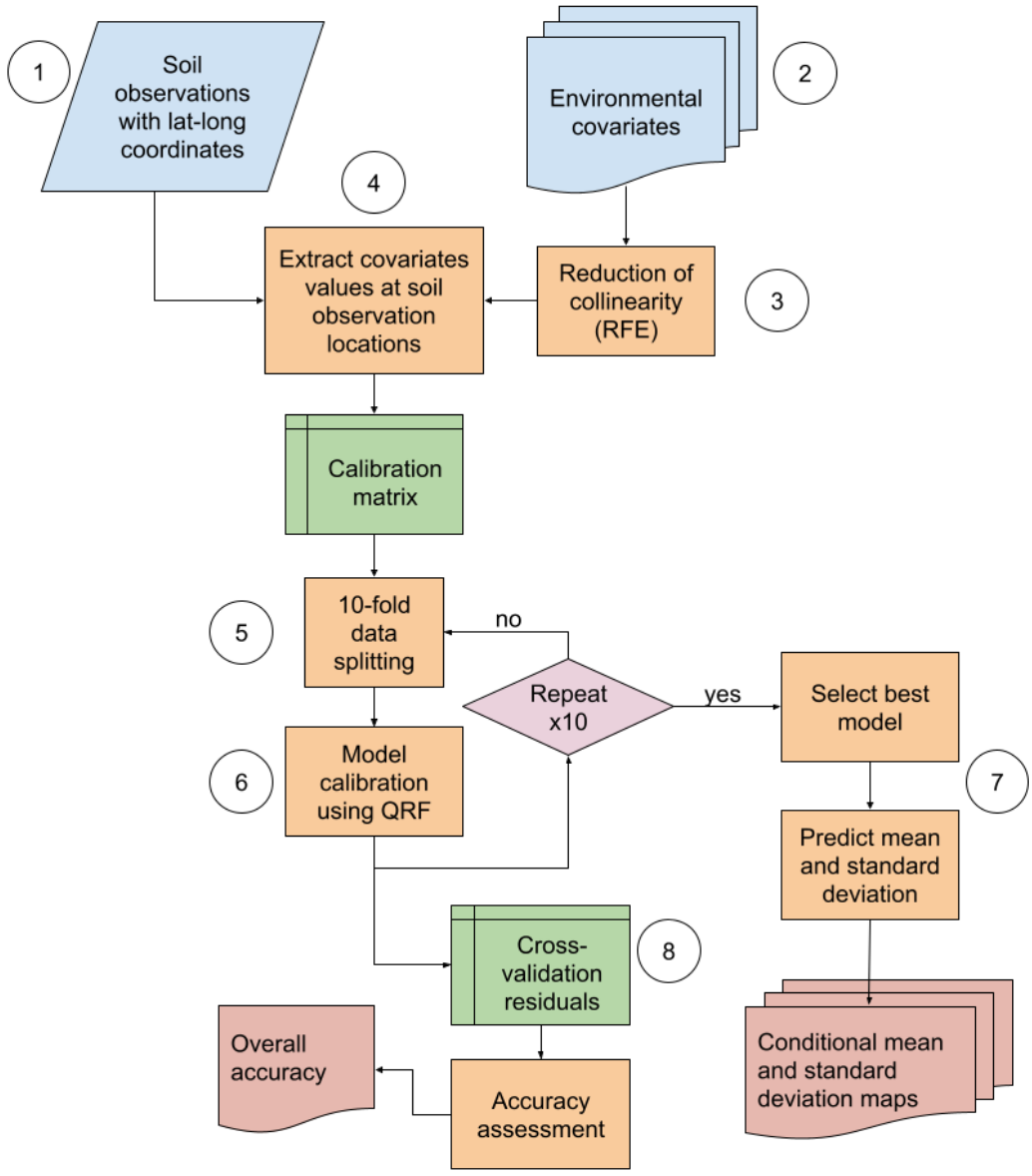


Figure 4.1: Digital soil mapping approach for point-support data. Circles are the steps.

Chapter 5

Arranging soil data in R

In this chapter, the datasets used in this technical manual are presented. They were adapted for educational purposes by the GSP Secretariat in order to best explain the way towards generating all necessary layers of the GSNmap initiative. Step-by-step instructions are given on how to carry out necessary steps in *RStudio* in order to be able to work with your soil data and generate maps. Instructions are given on how to:

1. Generate user-defined variables,
2. Set the working directory and load necessary packages,
3. Import national data to *RStudio*
4. Handle data (select, filter useful columns, etc.)

Thus, the instructions also serve as a continuation of the basic introduction to the functioning of **R** and *RStudio* given in Chapter 3. Still, in case for further information there is a vast amount of websites that offer help and or information on **R** and *RStudio*.

5.1 Study area and training material

The study area is located in the southeast of the Pampas Region, in Argentina, from the foothills of the Ventania and Tandilia hill systems, until the southern

coasts of the Buenos Aires Province. To illustrate the different processes of this Technical Manual, we use three datasets from this region:

- Georeferenced topsoil data
  - Chemical soil properties
  - Physical soil properties
- Soil profile data

5.1.1 Georeferenced topsoil data

These data were collected in 2011 by the National Institute of Agriculture Technology and Faculty of Agricultural Science of the National University of Mar del Plata (Unidad Integrada INTA-FCA) to map the status of soil nutrients in the Argentinian Pampas (Sainz Rozas *et al.*, 2019). The modified dataset is derived from a subset of 118 locations and covers the target depth of 0-30 cm. It is structured in two different spreadsheets that contain *soil chemical properties* (soil_chem_data030.csv) and *soil physical properties* (soil_phys_data030.csv). All datasets are located in the `01-Data` folder in the training material `Digital-Soil-Mapping` folder. Soil chemical properties spreadsheet data from laboratories with point coordinates (lat/long) together with data on available Phosphorus (p_bray, in ppm), available Potassium (k, in ppm), and total nitrogen (tn, in Percent) (see Table 5.1).

The spreadsheet with soil physical data contains data on soil texture for clay (clay_0_30, in g/kg), silt (silt_0_30, in g/kg), and sand (sand_0_30, in g/kg) (see Table 5.2).

The distribution of points is shown in the following map for available Phosphorus values as points. This dataset is used in Chapter 8 for mapping.

```
library(tidyverse)
library(sf)
library(mapview)

mapviewOptions(fgb = FALSE)
data <- read_csv("Digital-Soil-Mapping/01-Data/soil_chem_data030.csv")
s <- st_as_sf(data, coords = c("x", "y"), crs = 4326)
mapview(s, zcol = "p_bray", cex = 2.5, lwd = 0)
```

Table 5.1: Dataset with coordinates for chemical soil properties.

LabID	x	y	p_bray	k	tn
51	-61.51282	-37.37646	20.40	852.17	0.22
60	-57.84725	-37.85136	10.52	769.55	0.30
64	-58.87620	-38.54000	15.87	992.41	0.27
67	-60.30394	-38.45300	20.85	740.24	0.18
68	-60.39772	-38.51567	13.54	724.77	0.17
69	-60.41442	-38.52914	46.17	699.03	0.13
74	-60.00556	-38.76500	20.94	518.58	0.23
75	-60.10750	-38.76472	26.82	450.17	0.24
77	-60.17139	-38.79278	22.56	858.80	0.19
78	-60.03111	-38.74611	20.09	662.91	0.20

Only the ten first rows are shown.

Table 5.2: Dataset with coordinates for physical soil properties.

ProfID	x	y	clay_0_30	sand_0_30	silt_0_30
154	-58.67430	-38.20796	259.79	410	330.21
197	-60.45918	-38.36285	251.05	400	348.95
262	-58.86694	-38.42194	213.04	480	306.96
2702	-58.02222	-37.82167	259.71	430	310.29
2706	-57.91861	-37.95444	265.08	400	334.92
2709	-60.47222	-36.67778	323.92	310	366.08
2710	-60.22856	-36.69115	274.30	240	485.70
2711	-60.45076	-36.84394	234.67	540	225.33
2712	-60.42631	-36.94468	293.42	310	396.58
2714	-59.27717	-36.95655	262.34	460	277.66

Only the ten first rows are shown.

Table 5.3: Soil profile dataset.

id_prof	id_hor	x	y	top	bottom	ph_h2o	k	soc	bd
51	28706	-60.35188	-38.80600	0	15	6	1.5	2.6	NA
51	28707	-60.35188	-38.80600	15	25	6	1.7	2.5	NA
51	28708	-60.35188	-38.80600	25	52	6	0.8	1.3	NA
51	28709	-60.35188	-38.80600	52	57	NA	NA	NA	NA
154	28425	-58.67430	-38.20796	0	14	6	2.2	3.6	NA
154	28426	-58.67430	-38.20796	14	26	6	1.9	2.8	NA
154	28427	-58.67430	-38.20796	26	44	6	2.5	1.1	NA
154	28428	-58.67430	-38.20796	44	56	7	2.2	0.5	NA
154	28429	-58.67430	-38.20796	56	105	6	1.8	0.2	NA
197	28588	-60.45918	-38.36285	0	13	6	2.8	3.4	NA

Only ten rows are shown.

5.1.2 Soil profile data

Finally, the third dataset belongs to the Soil Information System of Argentina (SISINTA, Olmedo, Rodriguez and Angelini (2017)) which contains soil profiles collected from the sixties to recently years for soil survey purposes. The data can be fetched using the package SISINTAR. Table 5.3 shows a subset of the data, and the map presents the distribution of soil profiles for the study area. Soil profile data consists of measurements of soil organic carbon (soc, in Percent), soil pH (ph_h2o), available Potassium (k), bulk density (bd, in g/cm^3), cation exchange capacity (cec, in cmol$_c$/100g). This dataset is used in this chapter to illustrate the preprocessing steps required for data that come from soil profiles.

```
library(tidyverse)
library(sf)
library(mapview)
mapviewOptions(fgb = FALSE)
data <- read_csv("Digital-Soil-Mapping/01-Data/soil_profile_data.csv")
s <- data %>% filter(top==0)
s <- st_as_sf(s, coords = c("x", "y"), crs = 4326)
mapview(s, zcol = "k", cex = 2.5, lwd = 0)
```

Table 5.4: Example format of a database.

Profile ID	Horizon ID	Lat	Long	Year	Top	Bottom	cec	ph	clay	silt
1	1_1	12.12346	1.123456	2018	0	20	15	6.5	35	58
1	1_2	12.12346	1.123456	2018	20	40	19	7.1	42	48
2	2_1	23.12346	2.123456	2019	0	30	14	5.5	12	53

Note:
Profile ID = unique profile identifier, Horizon ID = unique layer identifier, Lat = latitude in d
= longitude in decimal degrees, Year = sampling year, Top = upper limit of the layer in cm, B
of the layer in cm, cec = Cation Exchange Capacity (cmol_c/kg), ph = pH in water, clay =
silt = Silt (g/100g soil), sand = Sand ((g/100g soil), soc = Soil Organic Carbon (g/100g soil)
(g/cm3).

5.2 Format requirements of soil data

Soil data consist of measurement at a specific geographical location, time and soil depth. Therefore, it is necessary to arrange the data following the format shown in Table 5.4.

5.3 Pre-processing steps

Soil data is often arranged in a different way which requires specific pre-processing steps to reach the format. On the way towards a formatted database, common issues such as, arranging the data format, fixing soil horizon depth consistency, detecting unusual soil property measurements, can be solved. Here, common issues and examples are given on how to carry out some basic data handling steps in *RStudio*.

5.3.1 Set the scene (set working directory, packages, load data)

So, let's open *RStudio*. Whenever starting to work on a project or task, it is necessary to set the *working directory* (WD). The WD is the folder path that is used by **R** to save the output, for instance a plot or a table that was generated

while working in **R**. Thus, the WD is central since it dictates where the files and calculations can be found afterwards. As it is so important, there are multiple ways of setting the WD. One option is to right click on 'Session' menu > 'Set working directory ...' and select either 'To Source File Location' (then the WD corresponds to the file path where the Script is saved to) or 'Choose Directory...'. Then, the user can browse to the folder that should be the WD.

In this manual we propose an alternative way that allows for more customization and flexibility since sometimes multiple WDs are needed to for instance save the final map in a different folder than the covariates. Since the file paths differ depending on where you stored the file on your computer, it is crucial to identify the correct file path. This can be done by accessing the *file explorer*. There you can browse to your training material folder and then right-click on the bar highlighted in red in the Figure 5.1.

The file path will appear with the following format: `C:\Users\GSNmap-TM\Digital-Soil-Mapping`
In order to enable **R** to read this as file path, it is necessary to replace the `\` by `/`. The resulting file path should look similar to this one: `C:/Users/GSNmap-TM/Digital-Soil-Mapping`. Once this is done, we can assign the file path that represents the WD file path to an **R** object. This is done by defining a character value (in this case the file path) on the right side of the arrow (`<-`) and name the **R** object on the left side (wd) (see code). Once this is done we use the function `setwd()` to set the WD to the file path that is specified in the object `wd`.

```
# 0 - User-defined variables ========================================================
wd <- 'C:/Users/hp/Documents/GitHub/Digital-Soil-Mapping'
#wd <- "C:/GIT/Digital-Soil-Mapping"

# 1 - Set working directory and load necessary packages =========================
setwd(wd) # change the path accordingly
```

Next to in-built base R functions, there is a vast amount of so-called packages that extend the functionalities of **R** and allow the use of **R** for a broad range of purposes. For data handling and management, the `tidyverse` package and its dependencies offer a great help. To load packages into the *RStudio* session, the `library` function is used. However, if the package is not installed, it is necessary to use the `install.packages` function first.

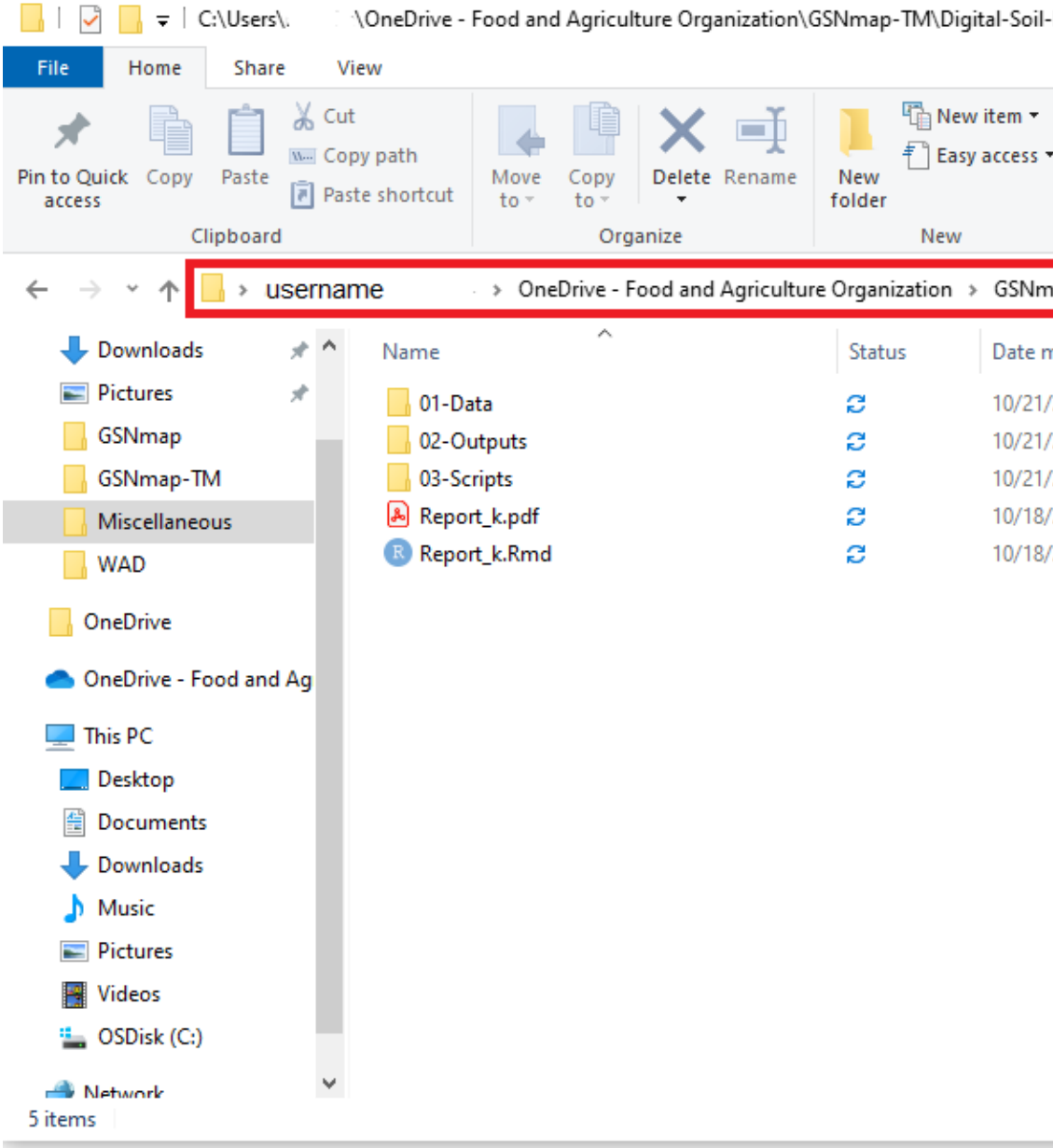


Figure 5.1: Get file path from file explorer.

```r
#install.packages(tidyverse)
library(readxl)
library(tidyverse)
library(dplyr)

# load in data
data <- read_csv("Digital-Soil-Mapping/01-Data/soil_chem_data030.csv")
```

```
## Rows: 119 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): LabID
## dbl (5): x, y, p_bray, k, tn
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this messa
```

```r
head(data)
```

```
## # A tibble: 6 x 6
##   LabID     x     y p_bray     k    tn
##   <chr> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 51    -61.5 -37.4   20.4  852. 0.225
## 2 60    -57.8 -37.9   10.5  770. 0.301
## 3 64    -58.9 -38.5   15.9  992. 0.266
## 4 67    -60.3 -38.5   20.8  740. 0.179
## 5 68    -60.4 -38.5   13.5  725. 0.168
## 6 69    -60.4 -38.5   46.2  699. 0.129
```

5.3.2 Basic data handling operations

In this section, basic operations with data in **R** are covered. It is explained how to select columns, filter observations/rows by certain values, remove missing values (NAs), how to rename columns and finally how to check the structure and classes of the whole dataframe or specific columns. The loaded dataset may comprise columns that are not of interest for the specific task you are working

on. Therefore, it is recommendable to select the columns that are relevant to keep your working environment in *RStudio* clean. In the following example we specify the dataframe we want to select columns from and then link the dataset to the `select` function of the tidyverse to select only the coordinates and the ID column. Then, we store the selected columns in a new object called locations.

```
# Select columns
locations <- data %>% dplyr::select(LabID, x, y)
head(locations)
```

```
## # A tibble: 6 x 3
##   LabID     x     y
##   <chr> <dbl> <dbl>
## 1 51    -61.5 -37.4
## 2 60    -57.8 -37.9
## 3 64    -58.9 -38.5
## 4 67    -60.3 -38.5
## 5 68    -60.4 -38.5
## 6 69    -60.4 -38.5
```

Another important operation is to filter by certain row values. For that, we can use the `filter` function of the tidyverse. Here, we want to filter to have only samples that are located below -38 degrees latitude (y). To do this, we can follow the same syntax as in the example above

```
# Filter
south_locations <- data %>% filter(y <= -38)
head(south_locations)
```

```
## # A tibble: 6 x 6
##   LabID     x     y p_bray     k    tn
##   <chr> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 64    -58.9 -38.5   15.9  992. 0.266
## 2 67    -60.3 -38.5   20.8  740. 0.179
## 3 68    -60.4 -38.5   13.5  725. 0.168
## 4 69    -60.4 -38.5   46.2  699. 0.129
## 5 74    -60.0 -38.8   20.9  519. 0.229
## 6 75    -60.1 -38.8   26.8  450. 0.235
```

One key operation is to rename column names. This is highly recommended before you start running the scripts explained in the following chapters. **R** is case-sensitive. This means that it matters whether you write `ph`, `PH`, or `pH`. Therefore, it is of utmost importance to be consistent and aware of any typos. To rename columns, it may be of interest to know how the columns are named. This can be checked by the `names()` function. Let's say we want to rename the "LabID" column and the "p_bray" column to "ID" and "p" in the south_locations dataframe with base R. The way to do this, is to assign the names of the columns of the dataframe to a vector (here called `names`) and select the name of a specific column in the vector by using squared brackets `[]` and a number that indicates the position of the column in the vector. Then, you can replace it with another name. Alternatively, one can follow the already known syntax of the tidyverse. Here, we rename three columns of the locations dataset using the `rename` function. Note that the new name is specified on the left side of the equal sign and the name to be replaced on the right side.

```
#check names
names(south_locations)
```

```
## [1] "LabID"  "x"      "y"      "p_bray" "k"      "tn"
```

```
names <- names(south_locations)
# renaming - base R option
names[1] <- "ID"
names[6] <- "k"
names(south_locations)
```

```
## [1] "LabID"  "x"      "y"      "p_bray" "k"      "tn"
```

```
# renaming - tidyverse option
names(locations)
```

```
## [1] "LabID" "x"     "y"
```

```
locations1 <- locations %>% rename(
  ID = LabID,
```

```
  long = x,
  lat = y
)
names(locations1)
```

```
## [1] "ID"   "long" "lat"
```

Another important operation when working with soil data is to remove NA values. NA values are empty cells that for instance do not contain coordinates. In the following case, we are going to check whether the locations dataset has NA values in the x column. To this end, we can use the `is.na` function. This function returns logical (TRUE or FALSE) values. Based on these values, any NA observations that have NA values in the x column can be removed from the dataset. For that we use again squared brackets `[]` and the logical operator `!` that equals `is not` to select all observations of locations that don't have NA values in the x column.

```
# identify NAs
is.na(locations$x)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
locations <- locations[!is.na(locations$x),]
head(locations)
```

```
## # A tibble: 6 x 3
##   LabID     x     y
##   <chr> <dbl> <dbl>
```

```
## 1 51    -61.5 -37.4
## 2 60    -57.8 -37.9
## 3 64    -58.9 -38.5
## 4 67    -60.3 -38.5
## 5 68    -60.4 -38.5
## 6 69    -60.4 -38.5
```

Finally, another common source of error arises from wrongly classified columns. For instance, if the column containing the pH measurements is not numeric but is classified as character (means text). To check which classes were assigned to each column after reading in the data, it is recommendable to use `str()` and `summary()` to get an overview of the dataset. In case the variables are not assigned to the correct class one can use the `as.numeric` or `as.character` functions to convert the respective column. However, in the present case all variables are assigned correctly. If it is not the case with your soil data, there might be issues with some observations, e.g. text values in certain observations.

```
# check for class
str(data)
```

```
## spc_tbl_ [119 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ LabID : chr [1:119] "51" "60" "64" "67" ...
##  $ x     : num [1:119] -61.5 -57.8 -58.9 -60.3 -60.4 ...
##  $ y     : num [1:119] -37.4 -37.9 -38.5 -38.5 -38.5 ...
##  $ p_bray: num [1:119] 20.4 10.5 15.9 20.8 13.5 ...
##  $ k     : num [1:119] 852 770 992 740 725 ...
##  $ tn    : num [1:119] 0.225 0.301 0.266 0.179 0.168 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   LabID = col_character(),
##   ..   x = col_double(),
##   ..   y = col_double(),
##   ..   p_bray = col_double(),
##   ..   k = col_double(),
##   ..   tn = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
summary(data)
```

```
##     LabID                x                y              p_bray
##  Length:119         Min.   :-61.72   Min.   :-38.79   Min.   : 4.160
##  Class :character   1st Qu.:-60.49   1st Qu.:-38.45   1st Qu.: 9.798
##  Mode  :character   Median :-59.84   Median :-38.08   Median :13.545
##                     Mean   :-59.74   Mean   :-38.02   Mean   :15.117
##                     3rd Qu.:-58.78   3rd Qu.:-37.67   3rd Qu.:17.723
##                     Max.   :-57.63   Max.   :-36.72   Max.   :54.561
##        k                tn
##  Min.   : 216.9   Min.   :0.09382
##  1st Qu.: 523.9   1st Qu.:0.19910
##  Median : 652.3   Median :0.23984
##  Mean   : 656.8   Mean   :0.23421
##  3rd Qu.: 765.8   3rd Qu.:0.26955
##  Max.   :1103.0   Max.   :0.33781
```

```
# change class of columns

# to numeric
#data$ph <- as.numeric(data$ph)

# to character
#data$LabID <- as.character(data$LabID)
```

For further guidance and more in-depth techniques to administer and handle soil data in **R**, it is recommended to check the GitHub repository on soil database management of the GSP: FAO-GSP Soil DB. There, not only training data but also extensive example codes are available. For now, we continue working with the example dataset and assume that the dataset you are using complies with the format specified at the beginning.

Chapter 6

Step 1: soil data preparation

This chapter builds on the previous one as it requires basic understanding of the data handling using **R**. From this point onwards, the steps base on each other and are needed to complete the mapping process. The instructions covered in this chapter provide step-by-step instructions on the following items:

1. Perform a quality check of the data
2. Estimate bulk density using PTF
3. Harmonize soil layers (using splines)
4. Plot and save the formatted soil data

6.1 Load national data

As specified in the previous Chapter in regards to pre-processing, at first the necessary steps in *RStudio* are taken: set the working directory and load the necessary R packages. Note that there are many ways to install packages besides the most common way using the function `install.packages()`. For instance, to install the `terra` package, one has to write `install.packages("terra")`. This installs the package from CRAN. However, there are a few exceptions where development versions of R packages are required. In these instances additional packages such as `devtools` or `remotes` are needed (see example in

code below). These packages are then able to install packages from for instance GitHub repositories.

```
# 0 - User-defined variables ===================================================
#wd <- 'C:/Users/luottoi/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
#wd <- "C:/GIT/GSNmap-TM/Digital-Soil-Mapping"

# 1 - Set working directory and load necessary packages ========================
setwd(wd) # change the path accordingly

library(tidyverse) # for data management and reshaping
library(readxl) # for importing excel files
library(mapview) # for seeing the profiles in a map
library(sf) # to manage spatial data (shp vectors)
library(aqp) # for soil profile data
#install.packages("devtools")
#devtools::install_bitbucket("brendo1001/ithir/pkg") #install ithir package
library(mpspline2) # for horizon harmonization
```

The next step is to load the national soil data into *R Studio*. For that, it is recommendable to have the data in either Microsoft Excel format (.xlsx) or as comma separated value table (.csv). In both cases, each row represents a sample (or horizon) and each column represents a variable. Then, the datasets can be loaded from the specified folder using the respective functions specified in the code below. It is noteworthy that in **R** datasets also need to be assigned to a user-defined variable in order to be saved in the "global environment".

In this example, the three different data tables are loaded into *RStudio*. The soil profile database of SISINTA (`hor`), the chemical (`chem`) and physical soil property tables (`phys`). After reading in the file, the package `tidyverse` comes into play. By using the `select()` and `unique()` functions, the user can select only the necessary columns from the table and ensure that no duplicates are included. At this point it may be necessary to rename certain columns, as shown for the Profile and Horizon ID columns in the code below. Finally, every time new datasets are loaded into *RStudio*, it is recommendable to check the data. Using the `summary()` function, users can see the class of each variable (= column) and descriptive statistics (for numerical variables). Classes are 'character' (`chr`) for text, integer (`int`) for whole numbers, and numeric (`num`) for numeric variables.

```r
# 2 - Import national data ============================================
# Save your national soil dataset in the data folder /01-Data as a .csv file or
# as a .xlsx file

## 2.1 - for .xlsx files ----------------------------------------------------
# Import horizon data
# hor <- read_excel("01-Data/soil_data.xlsx", sheet = 2)
# # Import site-level data
# site <- read_excel("01-Data/soil_data.xlsx", sheet = 1)
# chem <- read_excel("01-Data/soil_data.xlsx", sheet = 2)
# phys <- read_excel("01-Data/soil_data.xlsx", sheet = 3)


## 2.2 - for .csv files -----------------------------------------------------
# Import horizon data
hor <- read_csv(file = "Digital-Soil-Mapping/01-Data/soil_profile_data.csv")
chem <- read_csv(file = "Digital-Soil-Mapping/01-Data/soil_chem_data030.csv")
phys <- read_csv(file = "Digital-Soil-Mapping/01-Data/soil_phys_data030.csv")

site <- select(hor, id_prof, x, y) %>% unique()
hor <- select(hor, id_prof, id_hor, top:cec)

# change names of key columns
names(site)
```

```
## [1] "id_prof" "x"       "y"
```

```r
names(site)[1] <- "ProfID"
names(hor)
```

```
## [1] "id_prof" "id_hor"  "top"     "bottom"  "ph_h2o"  "k"       "soc"
## [8] "bd"      "cec"
```

```r
names(hor)[1] <- "ProfID"
names(hor)[2] <- "HorID"
# scan the data
summary(site)
```

```
##      ProfID           x                y
##  Min.   :  51   Min.   :-61.64   Min.   :-38.81
##  1st Qu.:6511   1st Qu.:-60.40   1st Qu.:-37.93
##  Median :7092   Median :-59.28   Median :-37.54
##  Mean   :6169   Mean   :-59.40   Mean   :-37.54
##  3rd Qu.:7383   3rd Qu.:-58.40   3rd Qu.:-37.10
##  Max.   :8128   Max.   :-57.55   Max.   :-36.56
```

```
summary(hor)
```

```
##      ProfID         HorID            top             bottom
##  Min.   :  51   Min.   :12230   Min.   :  0.00   Min.   :  5.00
##  1st Qu.:6512   1st Qu.:29161   1st Qu.: 15.00   1st Qu.: 28.00
##  Median :6948   Median :31464   Median : 35.00   Median : 55.00
##  Mean   :6166   Mean   :28491   Mean   : 42.67   Mean   : 62.14
##  3rd Qu.:7385   3rd Qu.:33766   3rd Qu.: 65.00   3rd Qu.: 88.00
##  Max.   :8128   Max.   :37674   Max.   :190.00   Max.   :230.00
##
##      ph_h2o            k               soc              bd
##  Min.   : 5.00   Min.   : 0.200   Min.   : 0.020   Min.   :0.87
##  1st Qu.: 6.70   1st Qu.: 1.400   1st Qu.: 0.250   1st Qu.:1.16
##  Median : 7.40   Median : 1.900   Median : 0.720   Median :1.26
##  Mean   : 7.59   Mean   : 1.994   Mean   : 1.457   Mean   :1.26
##  3rd Qu.: 8.50   3rd Qu.: 2.500   3rd Qu.: 2.360   3rd Qu.:1.38
##  Max.   :10.30   Max.   :16.800   Max.   :19.000   Max.   :1.49
##  NA's   :318     NA's   :331      NA's   :358      NA's   :1804
##       cec
##  Min.   : 2.40
##  1st Qu.:19.70
##  Median :24.90
##  Mean   :25.38
##  3rd Qu.:30.30
##  Max.   :66.60
##  NA's   :354
```

The selection of useful columns is very important since it ensures that users keep a good overview and a clean environment. Using the `select()` function, it is also possible to rename the variables right away (see code below).

```
# 3 - select useful columns ====================================================
## 3.1 - select columns ------------------------------------------------------
hor <- select(hor, ProfID, HorID, top, bottom, ph=ph_h2o, k, soc, bd, cec)
```

6.2 Data quality check

Datasets need to be checked for their quality as especially manually entered data is prone to mistakes such as typos or duplicates. A thorough quality check ensures that:

- all profiles have reasonable coordinates (within the area of interest);
- there are no duplicated profiles; and
- the depth logic within a profile is not violated.

To check the first point, the dataframe needs to be converted into a spatial object using the `st_as_sf()` function of the `sf` package. It is necessary to indicate the columns that contains latitude and longitude, as well as a coordinate reference system (CRS). We recommend WGS84 which corresponds to an EPSG code of 4326. However, locally more appropriate CRS can be found on the following website: https://epsg.io/. The `mapview()` command (from `mapview` package) offers the possibility to visualize the profile locations in an interactive map. Finally, the `filter()` function can be used to remove rows that contain profiles with wrong locations.

To visualize the profile locations, the soil data table was converted into a shapefile. Still, to check whether the database complies with the depth logic within each profile, it is necessary to convert the data table into a so-called soil profile collection that allows for very specific operations. These operations were bundled in the package `aqp` (AQP = Algorithms for Quantitative Pedology) (Beaudette, Roudier and O'Geen, 2013). With the first lines of code below, the dataset is converted into a soil profile collection and profiles and horizon tables are joined based on the site information. Now the profile collection can be visualised for any soil property. In this case, only the first 20 profiles are selected for the cation exchange capacity (CEC). Using the `checkHzDepthLogic()` function, users can assess that all profiles do not have gaps or overlaps of neighbouring horizons.

```
## 4.2 - Convert data into a Soil Profile Collection ----------------------------
library(aqp)
```

```
## This is aqp 1.42
```

```
##
## Attaching package: 'aqp'
```

```
## The following object is masked from 'package:plotly':
##
##     slice
```

```
## The following objects are masked from 'package:dplyr':
##
##     combine, slice
```

```
depths(hor) <- ProfID ~ top + bottom
hor@site$ProfID <- as.numeric(hor@site$ProfID)
site(hor) <- left_join(site(hor), site)
```

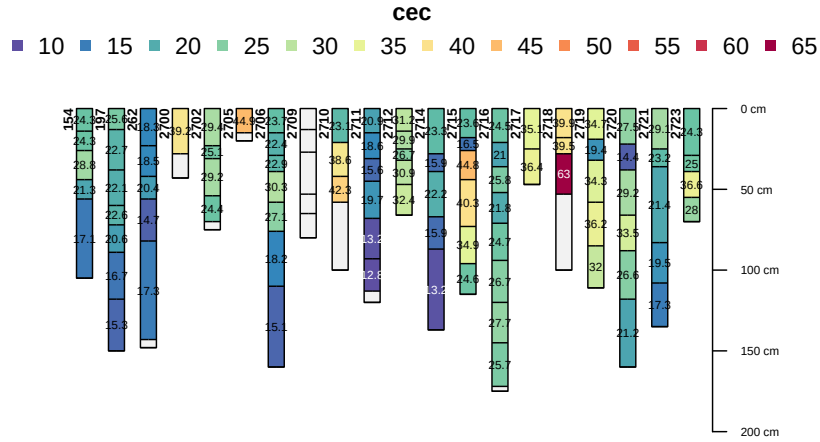
```
## Joining with `by = join_by(ProfID)`
```

```
profiles <- hor

profiles
```

```
## SoilProfileCollection with 357 profiles and 1813 horizons
## profile ID: ProfID  |  horizon ID: hzID
## Depth range: 5 - 230 cm
##
## ----- Horizons (6 / 1813 rows  |  10 / 10 columns) -----
## # A tibble: 6 x 10
##   ProfID hzID    top bottom HorID    ph     k   soc    bd   cec
##    <dbl> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1    154 1         0     14 28425   6     2.2  3.62    NA  24.3
```

```
## 2    154 2        14     26 28426   6     1.9  2.84    NA  24.3
## 3    154 3        26     44 28427   6.4   2.5  1.06    NA  28.8
## 4    154 4        44     56 28428   6.7   2.2  0.46    NA  21.3
## 5    154 5        56    105 28429   6.5   1.8  0.16    NA  17.1
## 6    197 6         0     13 28588   5.9   2.8  3.35    NA  25.6
## [... more horizons ...]
##
## ----- Sites (6 / 357 rows  |  3 / 3 columns) -----
## # A tibble: 6 x 3
##   ProfID     x     y
##    <dbl> <dbl> <dbl>
## 1    154 -58.7 -38.2
## 2    197 -60.5 -38.4
## 3    262 -58.9 -38.4
## 4   2700 -58.5 -37.7
## 5   2702 -58.0 -37.8
## 6   2705 -57.9 -37.9
## [... more sites ...]
##
## Spatial Data:
## [EMPTY]
```

```
## 4.3 - plot first 20 profiles using pH as color ------------------------------
plotSPC(x = profiles[1:20], name = "cec", color = "cec",
        name.style = "center-center")
```




```
## 4.4 - check data integrity ------------------------------------------------
# A valid profile is TRUE if all of the following criteria are false:
#    + depthLogic : boolean, errors related to depth logic
#    + sameDepth : boolean, errors related to same top/bottom depths
#    + missingDepth : boolean, NA in top / bottom depths
#    + overlapOrGap : boolean, gaps or overlap in adjacent horizons
aqp::checkHzDepthLogic(profiles)
```

```
##     ProfID valid depthLogic sameDepth missingDepth overlapOrGap
## 1      154  TRUE      FALSE     FALSE        FALSE        FALSE
## 2      197  TRUE      FALSE     FALSE        FALSE        FALSE
## 3      262  TRUE      FALSE     FALSE        FALSE        FALSE
## 4     2700  TRUE      FALSE     FALSE        FALSE        FALSE
## 5     2702  TRUE      FALSE     FALSE        FALSE        FALSE
## 6     2705  TRUE      FALSE     FALSE        FALSE        FALSE
## 7     2706  TRUE      FALSE     FALSE        FALSE        FALSE
## 8     2709  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 9     2710  TRUE      FALSE     FALSE        FALSE        FALSE
## 10    2711  TRUE      FALSE     FALSE        FALSE        FALSE
## 11    2712  TRUE      FALSE     FALSE        FALSE        FALSE
## 12    2714  TRUE      FALSE     FALSE        FALSE        FALSE
## 13    2715  TRUE      FALSE     FALSE        FALSE        FALSE
## 14    2716  TRUE      FALSE     FALSE        FALSE        FALSE
## 15    2717  TRUE      FALSE     FALSE        FALSE        FALSE
## 16    2718  TRUE      FALSE     FALSE        FALSE        FALSE
## 17    2719  TRUE      FALSE     FALSE        FALSE        FALSE
## 18    2720  TRUE      FALSE     FALSE        FALSE        FALSE
## 19    2721  TRUE      FALSE     FALSE        FALSE        FALSE
## 20    2723  TRUE      FALSE     FALSE        FALSE        FALSE
## 21    2725  TRUE      FALSE     FALSE        FALSE        FALSE
## 22    2726  TRUE      FALSE     FALSE        FALSE        FALSE
## 23    2761  TRUE      FALSE     FALSE        FALSE        FALSE
## 24    2762  TRUE      FALSE     FALSE        FALSE        FALSE
## 25    2763  TRUE      FALSE     FALSE        FALSE        FALSE
## 26    2764  TRUE      FALSE     FALSE        FALSE        FALSE
## 27    2767  TRUE      FALSE     FALSE        FALSE        FALSE
## 28    2768  TRUE      FALSE     FALSE        FALSE        FALSE
## 29    2769  TRUE      FALSE     FALSE        FALSE        FALSE
## 30    2770  TRUE      FALSE     FALSE        FALSE        FALSE
## 31    2771  TRUE      FALSE     FALSE        FALSE        FALSE
## 32    2772  TRUE      FALSE     FALSE        FALSE        FALSE
## 33    2773  TRUE      FALSE     FALSE        FALSE        FALSE
## 34    2774  TRUE      FALSE     FALSE        FALSE        FALSE
## 35    2775  TRUE      FALSE     FALSE        FALSE        FALSE
## 36    2796  TRUE      FALSE     FALSE        FALSE        FALSE
## 37    2797  TRUE      FALSE     FALSE        FALSE        FALSE
## 38    2798  TRUE      FALSE     FALSE        FALSE        FALSE
## 39    2799  TRUE      FALSE     FALSE        FALSE        FALSE
## 40    2800  TRUE      FALSE     FALSE        FALSE        FALSE
## 41    2801  TRUE      FALSE     FALSE        FALSE        FALSE
## 42    2802  TRUE      FALSE     FALSE        FALSE        FALSE
## 43    2803  TRUE      FALSE     FALSE        FALSE        FALSE
## 44    2804  TRUE      FALSE     FALSE        FALSE        FALSE
## 45    2805  TRUE      FALSE     FALSE        FALSE        FALSE
## 46    2806  TRUE      FALSE     FALSE        FALSE        FALSE
## 47    2807  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 48    2808  TRUE      FALSE     FALSE        FALSE        FALSE
## 49    2809  TRUE      FALSE     FALSE        FALSE        FALSE
## 50    2810  TRUE      FALSE     FALSE        FALSE        FALSE
## 51    2811  TRUE      FALSE     FALSE        FALSE        FALSE
## 52    2812  TRUE      FALSE     FALSE        FALSE        FALSE
## 53    2813  TRUE      FALSE     FALSE        FALSE        FALSE
## 54    2814  TRUE      FALSE     FALSE        FALSE        FALSE
## 55    2815  TRUE      FALSE     FALSE        FALSE        FALSE
## 56    2816  TRUE      FALSE     FALSE        FALSE        FALSE
## 57    2817  TRUE      FALSE     FALSE        FALSE        FALSE
## 58    2818  TRUE      FALSE     FALSE        FALSE        FALSE
## 59    2819  TRUE      FALSE     FALSE        FALSE        FALSE
## 60    2820  TRUE      FALSE     FALSE        FALSE        FALSE
## 61    2821  TRUE      FALSE     FALSE        FALSE        FALSE
## 62    2822  TRUE      FALSE     FALSE        FALSE        FALSE
## 63    2823  TRUE      FALSE     FALSE        FALSE        FALSE
## 64    2824  TRUE      FALSE     FALSE        FALSE        FALSE
## 65    2825  TRUE      FALSE     FALSE        FALSE        FALSE
## 66    2826  TRUE      FALSE     FALSE        FALSE        FALSE
## 67    2827  TRUE      FALSE     FALSE        FALSE        FALSE
## 68    2828  TRUE      FALSE     FALSE        FALSE        FALSE
## 69    2891  TRUE      FALSE     FALSE        FALSE        FALSE
## 70    2892  TRUE      FALSE     FALSE        FALSE        FALSE
## 71    2911  TRUE      FALSE     FALSE        FALSE        FALSE
## 72    2948  TRUE      FALSE     FALSE        FALSE        FALSE
## 73    2979  TRUE      FALSE     FALSE        FALSE        FALSE
## 74    3026  TRUE      FALSE     FALSE        FALSE        FALSE
## 75    3027  TRUE      FALSE     FALSE        FALSE        FALSE
## 76    3029  TRUE      FALSE     FALSE        FALSE        FALSE
## 77    3172  TRUE      FALSE     FALSE        FALSE        FALSE
## 78    3174  TRUE      FALSE     FALSE        FALSE        FALSE
## 79    3175  TRUE      FALSE     FALSE        FALSE        FALSE
## 80    3204  TRUE      FALSE     FALSE        FALSE        FALSE
## 81      51  TRUE      FALSE     FALSE        FALSE        FALSE
## 82    6180  TRUE      FALSE     FALSE        FALSE        FALSE
## 83    6181  TRUE      FALSE     FALSE        FALSE        FALSE
## 84    6182  TRUE      FALSE     FALSE        FALSE        FALSE
## 85    6506  TRUE      FALSE     FALSE        FALSE        FALSE
## 86    6507  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 87    6508  TRUE      FALSE     FALSE        FALSE        FALSE
## 88    6509  TRUE      FALSE     FALSE        FALSE        FALSE
## 89    6510  TRUE      FALSE     FALSE        FALSE        FALSE
## 90    6511  TRUE      FALSE     FALSE        FALSE        FALSE
## 91    6512  TRUE      FALSE     FALSE        FALSE        FALSE
## 92    6513  TRUE      FALSE     FALSE        FALSE        FALSE
## 93    6516  TRUE      FALSE     FALSE        FALSE        FALSE
## 94    6518  TRUE      FALSE     FALSE        FALSE        FALSE
## 95    6519  TRUE      FALSE     FALSE        FALSE        FALSE
## 96    6520  TRUE      FALSE     FALSE        FALSE        FALSE
## 97    6521  TRUE      FALSE     FALSE        FALSE        FALSE
## 98    6522  TRUE      FALSE     FALSE        FALSE        FALSE
## 99    6523  TRUE      FALSE     FALSE        FALSE        FALSE
## 100   6524  TRUE      FALSE     FALSE        FALSE        FALSE
## 101   6525  TRUE      FALSE     FALSE        FALSE        FALSE
## 102   6526  TRUE      FALSE     FALSE        FALSE        FALSE
## 103   6534  TRUE      FALSE     FALSE        FALSE        FALSE
## 104   6535  TRUE      FALSE     FALSE        FALSE        FALSE
## 105   6536  TRUE      FALSE     FALSE        FALSE        FALSE
## 106   6537  TRUE      FALSE     FALSE        FALSE        FALSE
## 107   6539  TRUE      FALSE     FALSE        FALSE        FALSE
## 108   6540  TRUE      FALSE     FALSE        FALSE        FALSE
## 109   6541  TRUE      FALSE     FALSE        FALSE        FALSE
## 110   6542  TRUE      FALSE     FALSE        FALSE        FALSE
## 111   6543  TRUE      FALSE     FALSE        FALSE        FALSE
## 112   6544  TRUE      FALSE     FALSE        FALSE        FALSE
## 113   6545  TRUE      FALSE     FALSE        FALSE        FALSE
## 114   6548  TRUE      FALSE     FALSE        FALSE        FALSE
## 115   6555  TRUE      FALSE     FALSE        FALSE        FALSE
## 116   6556  TRUE      FALSE     FALSE        FALSE        FALSE
## 117   6557  TRUE      FALSE     FALSE        FALSE        FALSE
## 118   6558  TRUE      FALSE     FALSE        FALSE        FALSE
## 119   6559  TRUE      FALSE     FALSE        FALSE        FALSE
## 120   6561  TRUE      FALSE     FALSE        FALSE        FALSE
## 121   6562  TRUE      FALSE     FALSE        FALSE        FALSE
## 122   6563  TRUE      FALSE     FALSE        FALSE        FALSE
## 123   6564  TRUE      FALSE     FALSE        FALSE        FALSE
## 124   6565  TRUE      FALSE     FALSE        FALSE        FALSE
## 125   6566 FALSE      FALSE     FALSE        FALSE         TRUE
```

```
## 126   6567  TRUE      FALSE     FALSE        FALSE        FALSE
## 127   6569  TRUE      FALSE     FALSE        FALSE        FALSE
## 128   6571  TRUE      FALSE     FALSE        FALSE        FALSE
## 129   6572  TRUE      FALSE     FALSE        FALSE        FALSE
## 130   6578  TRUE      FALSE     FALSE        FALSE        FALSE
## 131   6579  TRUE      FALSE     FALSE        FALSE        FALSE
## 132   6585  TRUE      FALSE     FALSE        FALSE        FALSE
## 133   6606  TRUE      FALSE     FALSE        FALSE        FALSE
## 134   6695  TRUE      FALSE     FALSE        FALSE        FALSE
## 135   6697  TRUE      FALSE     FALSE        FALSE        FALSE
## 136   6698  TRUE      FALSE     FALSE        FALSE        FALSE
## 137   6699  TRUE      FALSE     FALSE        FALSE        FALSE
## 138   6700  TRUE      FALSE     FALSE        FALSE        FALSE
## 139   6701  TRUE      FALSE     FALSE        FALSE        FALSE
## 140   6702  TRUE      FALSE     FALSE        FALSE        FALSE
## 141   6703  TRUE      FALSE     FALSE        FALSE        FALSE
## 142   6706  TRUE      FALSE     FALSE        FALSE        FALSE
## 143   6708  TRUE      FALSE     FALSE        FALSE        FALSE
## 144   6709  TRUE      FALSE     FALSE        FALSE        FALSE
## 145   6711  TRUE      FALSE     FALSE        FALSE        FALSE
## 146   6717  TRUE      FALSE     FALSE        FALSE        FALSE
## 147   6720  TRUE      FALSE     FALSE        FALSE        FALSE
## 148   6721  TRUE      FALSE     FALSE        FALSE        FALSE
## 149   6722  TRUE      FALSE     FALSE        FALSE        FALSE
## 150   6723  TRUE      FALSE     FALSE        FALSE        FALSE
## 151   6724  TRUE      FALSE     FALSE        FALSE        FALSE
## 152   6726  TRUE      FALSE     FALSE        FALSE        FALSE
## 153   6727  TRUE      FALSE     FALSE        FALSE        FALSE
## 154   6729  TRUE      FALSE     FALSE        FALSE        FALSE
## 155   6731  TRUE      FALSE     FALSE        FALSE        FALSE
## 156   6732  TRUE      FALSE     FALSE        FALSE        FALSE
## 157   6733  TRUE      FALSE     FALSE        FALSE        FALSE
## 158   6752  TRUE      FALSE     FALSE        FALSE        FALSE
## 159   6753  TRUE      FALSE     FALSE        FALSE        FALSE
## 160   6754  TRUE      FALSE     FALSE        FALSE        FALSE
## 161   6902  TRUE      FALSE     FALSE        FALSE        FALSE
## 162   6903  TRUE      FALSE     FALSE        FALSE        FALSE
## 163   6913  TRUE      FALSE     FALSE        FALSE        FALSE
## 164   6915  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 165   6916  TRUE      FALSE     FALSE        FALSE        FALSE
## 166   6921  TRUE      FALSE     FALSE        FALSE        FALSE
## 167   6922  TRUE      FALSE     FALSE        FALSE        FALSE
## 168   6923  TRUE      FALSE     FALSE        FALSE        FALSE
## 169   6924  TRUE      FALSE     FALSE        FALSE        FALSE
## 170   6947  TRUE      FALSE     FALSE        FALSE        FALSE
## 171   6948  TRUE      FALSE     FALSE        FALSE        FALSE
## 172   6950  TRUE      FALSE     FALSE        FALSE        FALSE
## 173   6958  TRUE      FALSE     FALSE        FALSE        FALSE
## 174   6959  TRUE      FALSE     FALSE        FALSE        FALSE
## 175   6960  TRUE      FALSE     FALSE        FALSE        FALSE
## 176   6961  TRUE      FALSE     FALSE        FALSE        FALSE
## 177   6962  TRUE      FALSE     FALSE        FALSE        FALSE
## 178   7002  TRUE      FALSE     FALSE        FALSE        FALSE
## 179   7092  TRUE      FALSE     FALSE        FALSE        FALSE
## 180   7093  TRUE      FALSE     FALSE        FALSE        FALSE
## 181   7094  TRUE      FALSE     FALSE        FALSE        FALSE
## 182   7123  TRUE      FALSE     FALSE        FALSE        FALSE
## 183   7124  TRUE      FALSE     FALSE        FALSE        FALSE
## 184   7125  TRUE      FALSE     FALSE        FALSE        FALSE
## 185   7127  TRUE      FALSE     FALSE        FALSE        FALSE
## 186   7131  TRUE      FALSE     FALSE        FALSE        FALSE
## 187   7132  TRUE      FALSE     FALSE        FALSE        FALSE
## 188   7133  TRUE      FALSE     FALSE        FALSE        FALSE
## 189   7134  TRUE      FALSE     FALSE        FALSE        FALSE
## 190   7136  TRUE      FALSE     FALSE        FALSE        FALSE
## 191   7137  TRUE      FALSE     FALSE        FALSE        FALSE
## 192   7138  TRUE      FALSE     FALSE        FALSE        FALSE
## 193   7139  TRUE      FALSE     FALSE        FALSE        FALSE
## 194   7140  TRUE      FALSE     FALSE        FALSE        FALSE
## 195   7141  TRUE      FALSE     FALSE        FALSE        FALSE
## 196   7142  TRUE      FALSE     FALSE        FALSE        FALSE
## 197   7145  TRUE      FALSE     FALSE        FALSE        FALSE
## 198   7153  TRUE      FALSE     FALSE        FALSE        FALSE
## 199   7154  TRUE      FALSE     FALSE        FALSE        FALSE
## 200   7155  TRUE      FALSE     FALSE        FALSE        FALSE
## 201   7156  TRUE      FALSE     FALSE        FALSE        FALSE
## 202   7157  TRUE      FALSE     FALSE        FALSE        FALSE
## 203   7158  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 204   7159  TRUE      FALSE     FALSE        FALSE        FALSE
## 205   7161  TRUE      FALSE     FALSE        FALSE        FALSE
## 206   7162  TRUE      FALSE     FALSE        FALSE        FALSE
## 207   7163  TRUE      FALSE     FALSE        FALSE        FALSE
## 208   7164  TRUE      FALSE     FALSE        FALSE        FALSE
## 209   7165  TRUE      FALSE     FALSE        FALSE        FALSE
## 210   7166  TRUE      FALSE     FALSE        FALSE        FALSE
## 211   7170  TRUE      FALSE     FALSE        FALSE        FALSE
## 212   7172  TRUE      FALSE     FALSE        FALSE        FALSE
## 213   7174  TRUE      FALSE     FALSE        FALSE        FALSE
## 214   7175  TRUE      FALSE     FALSE        FALSE        FALSE
## 215   7176  TRUE      FALSE     FALSE        FALSE        FALSE
## 216   7202  TRUE      FALSE     FALSE        FALSE        FALSE
## 217   7207  TRUE      FALSE     FALSE        FALSE        FALSE
## 218   7208  TRUE      FALSE     FALSE        FALSE        FALSE
## 219   7209  TRUE      FALSE     FALSE        FALSE        FALSE
## 220   7210  TRUE      FALSE     FALSE        FALSE        FALSE
## 221   7211  TRUE      FALSE     FALSE        FALSE        FALSE
## 222   7212  TRUE      FALSE     FALSE        FALSE        FALSE
## 223   7213  TRUE      FALSE     FALSE        FALSE        FALSE
## 224   7219  TRUE      FALSE     FALSE        FALSE        FALSE
## 225   7220  TRUE      FALSE     FALSE        FALSE        FALSE
## 226   7222  TRUE      FALSE     FALSE        FALSE        FALSE
## 227   7223  TRUE      FALSE     FALSE        FALSE        FALSE
## 228   7224  TRUE      FALSE     FALSE        FALSE        FALSE
## 229   7225  TRUE      FALSE     FALSE        FALSE        FALSE
## 230   7226  TRUE      FALSE     FALSE        FALSE        FALSE
## 231   7228  TRUE      FALSE     FALSE        FALSE        FALSE
## 232   7329  TRUE      FALSE     FALSE        FALSE        FALSE
## 233   7330  TRUE      FALSE     FALSE        FALSE        FALSE
## 234   7331  TRUE      FALSE     FALSE        FALSE        FALSE
## 235   7336  TRUE      FALSE     FALSE        FALSE        FALSE
## 236   7337  TRUE      FALSE     FALSE        FALSE        FALSE
## 237   7338  TRUE      FALSE     FALSE        FALSE        FALSE
## 238   7339  TRUE      FALSE     FALSE        FALSE        FALSE
## 239   7340  TRUE      FALSE     FALSE        FALSE        FALSE
## 240   7341  TRUE      FALSE     FALSE        FALSE        FALSE
## 241   7342  TRUE      FALSE     FALSE        FALSE        FALSE
## 242   7343  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 243   7344  TRUE      FALSE     FALSE        FALSE        FALSE
## 244   7345  TRUE      FALSE     FALSE        FALSE        FALSE
## 245   7346  TRUE      FALSE     FALSE        FALSE        FALSE
## 246   7349  TRUE      FALSE     FALSE        FALSE        FALSE
## 247   7350  TRUE      FALSE     FALSE        FALSE        FALSE
## 248   7351  TRUE      FALSE     FALSE        FALSE        FALSE
## 249   7352  TRUE      FALSE     FALSE        FALSE        FALSE
## 250   7353  TRUE      FALSE     FALSE        FALSE        FALSE
## 251   7354  TRUE      FALSE     FALSE        FALSE        FALSE
## 252   7355  TRUE      FALSE     FALSE        FALSE        FALSE
## 253   7356  TRUE      FALSE     FALSE        FALSE        FALSE
## 254   7359  TRUE      FALSE     FALSE        FALSE        FALSE
## 255   7360  TRUE      FALSE     FALSE        FALSE        FALSE
## 256   7361  TRUE      FALSE     FALSE        FALSE        FALSE
## 257   7362  TRUE      FALSE     FALSE        FALSE        FALSE
## 258   7363  TRUE      FALSE     FALSE        FALSE        FALSE
## 259   7364  TRUE      FALSE     FALSE        FALSE        FALSE
## 260   7365  TRUE      FALSE     FALSE        FALSE        FALSE
## 261   7367  TRUE      FALSE     FALSE        FALSE        FALSE
## 262   7368  TRUE      FALSE     FALSE        FALSE        FALSE
## 263   7374  TRUE      FALSE     FALSE        FALSE        FALSE
## 264   7376  TRUE      FALSE     FALSE        FALSE        FALSE
## 265   7378  TRUE      FALSE     FALSE        FALSE        FALSE
## 266   7379  TRUE      FALSE     FALSE        FALSE        FALSE
## 267   7382  TRUE      FALSE     FALSE        FALSE        FALSE
## 268   7383  TRUE      FALSE     FALSE        FALSE        FALSE
## 269   7384  TRUE      FALSE     FALSE        FALSE        FALSE
## 270   7385  TRUE      FALSE     FALSE        FALSE        FALSE
## 271   7386  TRUE      FALSE     FALSE        FALSE        FALSE
## 272   7387  TRUE      FALSE     FALSE        FALSE        FALSE
## 273   7388  TRUE      FALSE     FALSE        FALSE        FALSE
## 274   7389  TRUE      FALSE     FALSE        FALSE        FALSE
## 275   7390  TRUE      FALSE     FALSE        FALSE        FALSE
## 276   7391  TRUE      FALSE     FALSE        FALSE        FALSE
## 277   7392  TRUE      FALSE     FALSE        FALSE        FALSE
## 278   7393  TRUE      FALSE     FALSE        FALSE        FALSE
## 279   7394  TRUE      FALSE     FALSE        FALSE        FALSE
## 280   7395  TRUE      FALSE     FALSE        FALSE        FALSE
## 281   7408  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 282   7410 FALSE      FALSE     FALSE        FALSE         TRUE
## 283   7411  TRUE      FALSE     FALSE        FALSE        FALSE
## 284   7420  TRUE      FALSE     FALSE        FALSE        FALSE
## 285   7421  TRUE      FALSE     FALSE        FALSE        FALSE
## 286   7422  TRUE      FALSE     FALSE        FALSE        FALSE
## 287   7423  TRUE      FALSE     FALSE        FALSE        FALSE
## 288   7424  TRUE      FALSE     FALSE        FALSE        FALSE
## 289   7426  TRUE      FALSE     FALSE        FALSE        FALSE
## 290   7434  TRUE      FALSE     FALSE        FALSE        FALSE
## 291   7435  TRUE      FALSE     FALSE        FALSE        FALSE
## 292   7436  TRUE      FALSE     FALSE        FALSE        FALSE
## 293   7437  TRUE      FALSE     FALSE        FALSE        FALSE
## 294   7438  TRUE      FALSE     FALSE        FALSE        FALSE
## 295   7439  TRUE      FALSE     FALSE        FALSE        FALSE
## 296   7440  TRUE      FALSE     FALSE        FALSE        FALSE
## 297   7441  TRUE      FALSE     FALSE        FALSE        FALSE
## 298   7442  TRUE      FALSE     FALSE        FALSE        FALSE
## 299   7443  TRUE      FALSE     FALSE        FALSE        FALSE
## 300   7444  TRUE      FALSE     FALSE        FALSE        FALSE
## 301   7447  TRUE      FALSE     FALSE        FALSE        FALSE
## 302   7448  TRUE      FALSE     FALSE        FALSE        FALSE
## 303   7451  TRUE      FALSE     FALSE        FALSE        FALSE
## 304   7452  TRUE      FALSE     FALSE        FALSE        FALSE
## 305   7455  TRUE      FALSE     FALSE        FALSE        FALSE
## 306   7456  TRUE      FALSE     FALSE        FALSE        FALSE
## 307   7457  TRUE      FALSE     FALSE        FALSE        FALSE
## 308   7458  TRUE      FALSE     FALSE        FALSE        FALSE
## 309   7459  TRUE      FALSE     FALSE        FALSE        FALSE
## 310   7687  TRUE      FALSE     FALSE        FALSE        FALSE
## 311   7697  TRUE      FALSE     FALSE        FALSE        FALSE
## 312   7726  TRUE      FALSE     FALSE        FALSE        FALSE
## 313   7833  TRUE      FALSE     FALSE        FALSE        FALSE
## 314   7882  TRUE      FALSE     FALSE        FALSE        FALSE
## 315   7907  TRUE      FALSE     FALSE        FALSE        FALSE
## 316   7922  TRUE      FALSE     FALSE        FALSE        FALSE
## 317   7933  TRUE      FALSE     FALSE        FALSE        FALSE
## 318   7937  TRUE      FALSE     FALSE        FALSE        FALSE
## 319   7941  TRUE      FALSE     FALSE        FALSE        FALSE
## 320   7950  TRUE      FALSE     FALSE        FALSE        FALSE
```

```
## 321   7955  TRUE      FALSE     FALSE        FALSE        FALSE
## 322   7965  TRUE      FALSE     FALSE        FALSE        FALSE
## 323   7966  TRUE      FALSE     FALSE        FALSE        FALSE
## 324   7973  TRUE      FALSE     FALSE        FALSE        FALSE
## 325   7978  TRUE      FALSE     FALSE        FALSE        FALSE
## 326   7981  TRUE      FALSE     FALSE        FALSE        FALSE
## 327   7986  TRUE      FALSE     FALSE        FALSE        FALSE
## 328   7987  TRUE      FALSE     FALSE        FALSE        FALSE
## 329   7988  TRUE      FALSE     FALSE        FALSE        FALSE
## 330   7990  TRUE      FALSE     FALSE        FALSE        FALSE
## 331   7991  TRUE      FALSE     FALSE        FALSE        FALSE
## 332   7993  TRUE      FALSE     FALSE        FALSE        FALSE
## 333   7994  TRUE      FALSE     FALSE        FALSE        FALSE
## 334   7997  TRUE      FALSE     FALSE        FALSE        FALSE
## 335   8000  TRUE      FALSE     FALSE        FALSE        FALSE
## 336   8002 FALSE      FALSE     FALSE        FALSE         TRUE
## 337   8004  TRUE      FALSE     FALSE        FALSE        FALSE
## 338   8006  TRUE      FALSE     FALSE        FALSE        FALSE
## 339   8010  TRUE      FALSE     FALSE        FALSE        FALSE
## 340   8011  TRUE      FALSE     FALSE        FALSE        FALSE
## 341   8012  TRUE      FALSE     FALSE        FALSE        FALSE
## 342   8018  TRUE      FALSE     FALSE        FALSE        FALSE
## 343   8021  TRUE      FALSE     FALSE        FALSE        FALSE
## 344   8023  TRUE      FALSE     FALSE        FALSE        FALSE
## 345   8025  TRUE      FALSE     FALSE        FALSE        FALSE
## 346   8026  TRUE      FALSE     FALSE        FALSE        FALSE
## 347   8027  TRUE      FALSE     FALSE        FALSE        FALSE
## 348   8028  TRUE      FALSE     FALSE        FALSE        FALSE
## 349   8067  TRUE      FALSE     FALSE        FALSE        FALSE
## 350   8069  TRUE      FALSE     FALSE        FALSE        FALSE
## 351   8070  TRUE      FALSE     FALSE        FALSE        FALSE
## 352   8075  TRUE      FALSE     FALSE        FALSE        FALSE
## 353   8083  TRUE      FALSE     FALSE        FALSE        FALSE
## 354   8093  TRUE      FALSE     FALSE        FALSE        FALSE
## 355   8099  TRUE      FALSE     FALSE        FALSE        FALSE
## 356   8122  TRUE      FALSE     FALSE        FALSE        FALSE
## 357   8128  TRUE      FALSE     FALSE        FALSE        FALSE
```

```r
# Identify non-valid profiles
dl <- checkHzDepthLogic(profiles)
dl[dl$depthLogic==T | dl$sameDepth==T | dl$missingDepth==T | dl$overlapOrGap==T,"
```

```
## [1] 6566 7410 8002
```

If there are profiles that violate the depth logic rules (i.e. overlapping horizons), they can be selected and checked through the Profile ID. In the following step, only profiles with valid horizon logic are selected. Finally, the soil profile collection is re-converted to a dataframe. With this, the quality check is finished.

```r
# visualize some of these profiles by the pid
subset(profiles, grepl(6566, ProfID, ignore.case = TRUE))
```

```
## SoilProfileCollection with 1 profiles and 4 horizons
## profile ID: ProfID  |  horizon ID: hzID
## Depth range: 80 - 80 cm
##
## ----- Horizons (4 / 4 rows  |  10 / 10 columns) -----
## # A tibble: 4 x 10
##   ProfID hzID    top bottom HorID    ph     k   soc    bd   cec
##    <dbl> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   6566 643       0     16 29463   9.4   1.5  2.02    NA  14
## 2   6566 644      16     25 29464   9.5   1.2  0.91    NA  13
## 3   6566 645      26     55 29465   9.1   2.6  0.42    NA  22.7
## 4   6566 646      55     80 29466   9     2.2  0.19    NA  18.8
##
## ----- Sites (1 / 1 rows  |  3 / 3 columns) -----
## # A tibble: 1 x 3
##   ProfID     x     y
##    <dbl> <dbl> <dbl>
## 1   6566 -58.2 -37.9
##
## Spatial Data:
##      [,1]
## [1,]   NA
## CRS:  NA
```

```
subset(profiles, grepl(6915, ProfID, ignore.case = TRUE))
```

```
## SoilProfileCollection with 1 profiles and 7 horizons
## profile ID: ProfID  |  horizon ID: hzID
## Depth range: 140 - 140 cm
##
## ----- Horizons (6 / 7 rows  |  10 / 10 columns) -----
## # A tibble: 6 x 10
##   ProfID hzID    top bottom HorID    ph     k   soc    bd   cec
##    <dbl> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   6915 864       0      6 31278   6     2.8 15.8     NA  46.6
## 2   6915 865       6     14 31279   6.7   1.8  5.65    NA  31.8
## 3   6915 866      14     21 31280   7.5   0.9  2.02    NA  17.3
## 4   6915 867      21     35 31281   7.6   1    0.45    NA   9.1
## 5   6915 868      35     65 31282   8.2   2.3  0.41    NA  29.7
## 6   6915 869      65     87 31283   8.5   2.7  0.23    NA  33.1
## [... more horizons ...]
##
## ----- Sites (1 / 1 rows  |  3 / 3 columns) -----
## # A tibble: 1 x 3
##   ProfID     x     y
##    <dbl> <dbl> <dbl>
## 1   6915 -58.2 -37.6
##
## Spatial Data:
##      [,1]
## [1,]   NA
## CRS:  NA
```

```
subset(profiles, grepl(7726, ProfID, ignore.case = TRUE))
```

```
## SoilProfileCollection with 1 profiles and 5 horizons
## profile ID: ProfID  |  horizon ID: hzID
## Depth range: 155 - 155 cm
##
## ----- Horizons (5 / 5 rows  |  10 / 10 columns) -----
## # A tibble: 5 x 10
```

```
##   ProfID hzID    top bottom HorID    ph     k   soc    bd   cec
##    <dbl> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   7726 1581      0     27 35695   6.5   1.2 19       NA  27
## 2   7726 1582     27     38 35696   6.6   1.4  1.97    NA  24.8
## 3   7726 1583     38     70 35697   6.9   1.4  0.83    NA  29.7
## 4   7726 1584     70     97 35698   7.2   1.4  0.2     NA  18
## 5   7726 1585     97    155 35699   7.3   1.2 NA       NA  13.8
##
## ----- Sites (1 / 1 rows  |  3 / 3 columns) -----
## # A tibble: 1 x 3
##   ProfID     x     y
##    <dbl> <dbl> <dbl>
## 1   7726 -58.8 -37.3
##
## Spatial Data:
##      [,1]
## [1,]   NA
## CRS:  NA
```

```
## 4.5 - keep only valid profiles ------------------------------------------
clean_prof <- HzDepthLogicSubset(profiles)
```

```
## dropping profiles with invalid depth logic, see `metadata(x)$removed.profiles`
```

```
metadata(clean_prof)$removed.profiles
```

```
## [1] 6566 7410 8002
```

```
# write_rds(clean_prof, "01-Data/soilProfileCollection.rds")

## 4.6 convert soilProfileCollection to a table ---------------------------
dat <- left_join(clean_prof@site, clean_prof@horizons)
```

```
## Joining with `by = join_by(ProfID)`
```

```
dat <- select(dat, ProfID, HorID, x, y, top, bottom, ph:cec )
```

6.3 Calculation of pedo-transfer functions

In the cases of single-layer samples, which is common in sampling for nutrient determination, a locally calibrated pedotransfer function (PTF) should be applied. PTF will be also required to harmonise the laboratory methods. Experts from GLOSOLAN will provide advice in this regard.

Therefore, a customised function is introduced to our working environment. Users can write their own functions in **R**. This is often necessary when existing functions need to be customised or very specific calculations need to be performed. Functions greatly increase the efficiency of our code. For further information, it is recommendable to consult online resources on the topic (e.g. https://hbctraining.github.io/Intro-to-R/lessons/03_introR-functions-and-arguments.html).

The function `estimateBD` below calculates various PTFs that estimate BD. Which equation is used is determined by the user that has to choose one of the methods and also specify the SOC value of the respective horizon. The SOC values is first converted to OM by using the conversion factor of 1.724 and then inserted in the respective PTF. The return() command tells **R** which value to output.

```
# 5 - Estimate BD using pedotransfer functions ================================

# create the function with all PTF
method_names <- c("Saini1996", "Drew1973", "Jeffrey1979", "Grigal1989",
                  "Adams1973", "Honeyset_Ratkowsky1989")

estimateBD <- function(SOC=NULL, method=NULL) {
  OM <- SOC * 1.724
  BD <- switch(method,
               "Saini1996" = 1.62 - 0.06 * OM,
               "Drew1973" = 1 / (0.6268 + 0.0361 * OM),
               "Jeffrey1979" = 1.482 - 0.6786 * (log(OM)),
               "Grigal1989" = 0.669 + 0.941 * exp(1)^(-0.06 * OM),
```

```
                "Adams1973" = 100 / (OM / 0.244 + (100 - OM) / 2.65),
                "Honeyset_Ratkowsky1989" = 1 / (0.564 + 0.0556 * OM),
                stop("Invalid method specified.")
  )
  return(BD)
}
```

To apply the `estimateBD` function, first a test dataframe is created that includes the SOC values from the cleaned profile table as well as the respective existing BD measurements. The rows without values in one of the columns are excluded using the na.omit() function since we want to first evaluate the difference between estimated BDs and measured BDs. Now, the test dataframe is complemented by the estimated BDs derived from the PTFs for each method. To add new columns to an existing dataframe one has to write on the left-hand side of the arrow the name of the existing dataframe object (in this case BD_test), the dollar sign ($), and the name of the new column. Here, the names are given according to the used BD PTF.

```
## 5.1 - Select a pedotransfer function ----------------------------------------
# Create a test dataset with BD and SOC data
BD_test <- data.frame(SOC = dat$soc, BD_observed = dat$bd)

# Remove missing values
BD_test <- BD_test[complete.cases(BD_test),]
BD_test <- na.omit(BD_test)


# 5.2 - Estimate BLD for a subset using the pedotransfer functions ------------
for (i in method_names) {
  BD_test[[i]] <- estimateBD(BD_test$SOC, method = i)
}

# Print the resulting data frame
BD_test
```

```
##      SOC BD_observed Saini1996 Drew1973 Jeffrey1979 Grigal1989 Adams1973
## 367 1.64        1.16  1.450358 1.371991   0.7767016   1.463173  2.072261
## 373 2.38        1.26  1.373813 1.290451   0.5239880   1.404651  1.886665
```

```
## 374 0.50        1.49  1.568280 1.519946   1.5827721   1.562569  2.442399
## 376 2.05        1.14  1.407948 1.325584   0.6252763   1.430196  1.965153
## 377 0.44        1.36  1.574486 1.528622   1.6695198   1.568132  2.465577
## 386 0.75        1.43  1.542420 1.484831   1.3076235   1.539757  2.350336
## 387 0.75        1.38  1.542420 1.484831   1.3076235   1.539757  2.350336
## 389 3.20        0.87  1.288992 1.210718   0.3230883   1.344826  1.716329
## 394 1.11        1.25  1.505182 1.437024   1.0415837   1.507928  2.229331
##     Honeyset_Ratkowsky1989
## 367               1.386576
## 373               1.262414
## 374               1.634181
## 376               1.314922
## 377               1.649686
## 386               1.572597
## 387               1.572597
## 389               1.148456
## 394               1.491650
```

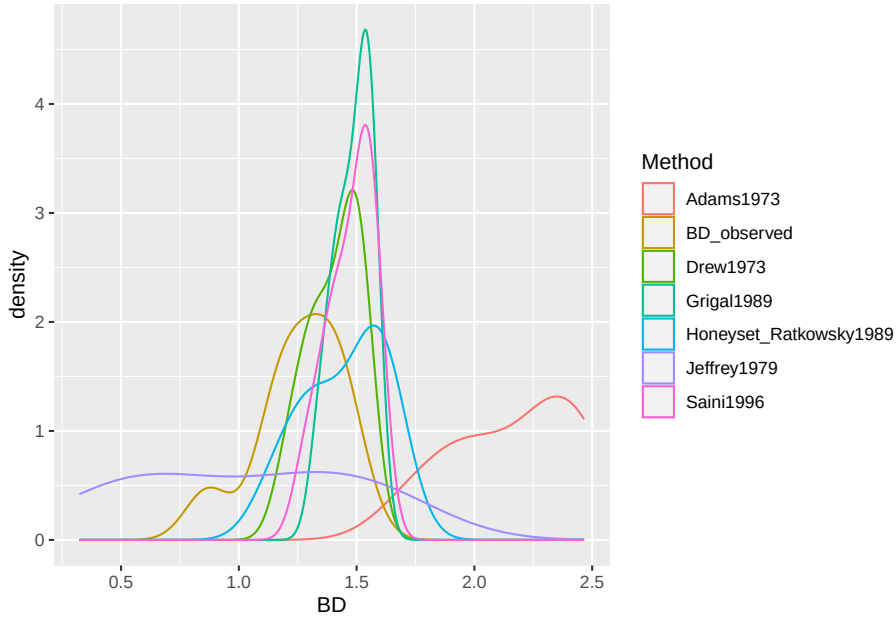
The calculated BDs can now be compared using the `summary()` function. However, a faster and more accessible approach is to plot the different bulk densities for comparison. In case you are not familiar with the `plot()` function and its respective commands, it is recommendable to check one of the many online learning resources such as https://intro2r.com/simple-base-r-plots.html. The plot shows us both measured and estimated BD values as differently coloured lines.

```
## 5.3 Compare results ------------------------------------------------------

# Observed values:
summary(BD_test)
```

```
##       SOC         BD_observed     Saini1996        Drew1973
##  Min.   :0.440   Min.   :0.87   Min.   :1.289   Min.   :1.211
##  1st Qu.:0.750   1st Qu.:1.16   1st Qu.:1.408   1st Qu.:1.326
##  Median :1.110   Median :1.26   Median :1.505   Median :1.437
##  Mean   :1.424   Mean   :1.26   Mean   :1.473   Mean   :1.406
##  3rd Qu.:2.050   3rd Qu.:1.38   3rd Qu.:1.542   3rd Qu.:1.485
##  Max.   :3.200   Max.   :1.49   Max.   :1.574   Max.   :1.529
```

```
##   Jeffrey1979       Grigal1989      Adams1973     Honeyset_Ratkowsky1989
##  Min.   :0.3231   Min.   :1.345   Min.   :1.716   Min.   :1.148
##  1st Qu.:0.6253   1st Qu.:1.430   1st Qu.:1.965   1st Qu.:1.315
##  Median :1.0416   Median :1.508   Median :2.229   Median :1.492
##  Mean   :1.0176   Mean   :1.485   Mean   :2.164   Mean   :1.448
##  3rd Qu.:1.3076   3rd Qu.:1.540   3rd Qu.:2.350   3rd Qu.:1.573
##  Max.   :1.6695   Max.   :1.568   Max.   :2.466   Max.   :1.650
```

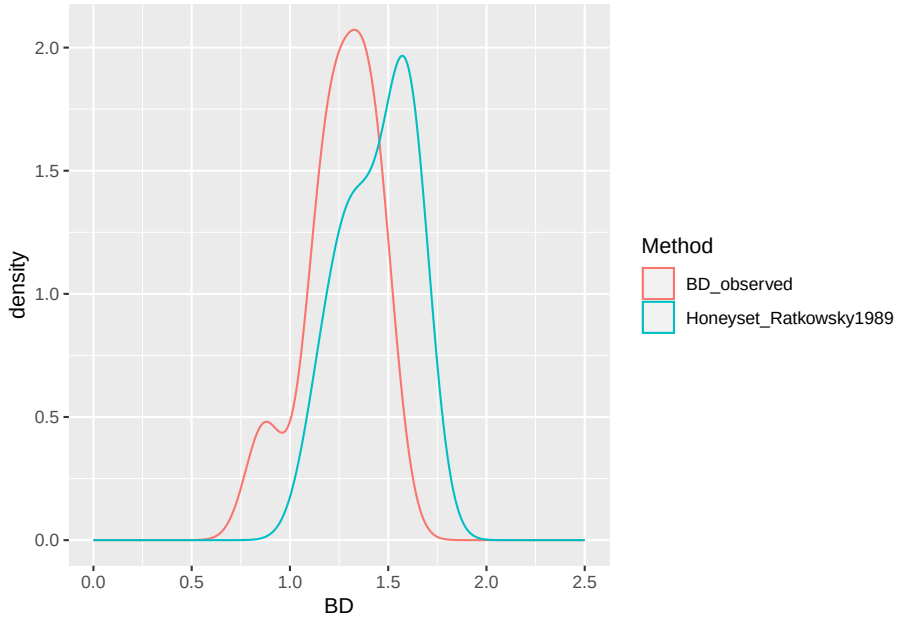
```r
# Compare data distributions for observed and predicted BLD
plot.bd <- BD_test %>%
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed", "Saini1996", "Drew1973", "Jeffrey1979",
                        "Grigal1989", "Adams1973", "Honeyset_Ratkowsky1989"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density()

plot.bd
```



```r
# Dymanic plot with plotly
ggplotly(plot.bd)
```

```r
ggplotly(plot.bd) %>%
  layout(hovermode = "x")
```

```r
# Plot the Selected function again
BD_test %>%
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed", "Honeyset_Ratkowsky1989"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density() + xlim(c(0,2.5))
```




```r
# Same dynamic plot
ggplotly(BD_test %>%
```

```
          select(-SOC) %>%
          pivot_longer(cols = c("BD_observed", "Honeyset_Ratkowsky1989"),
                       names_to = "Method", values_to = "BD") %>%
          ggplot(aes(x = BD, color = Method)) +
          geom_density() + xlim(c(0,2.5))) %>%
  layout(hovermode = "x")
```

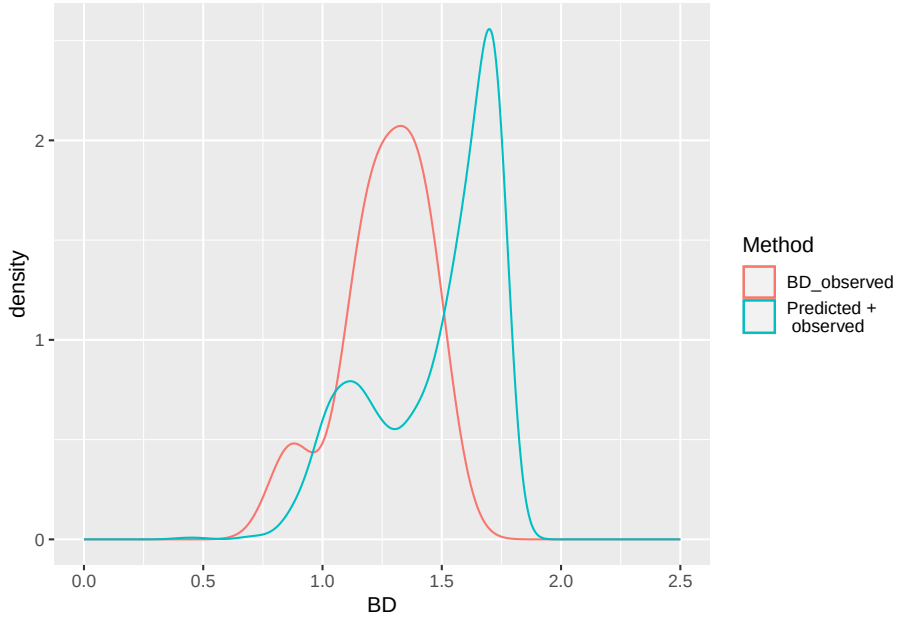
The PTF to be chosen for estimating the BD of the missing horizons should be the closest to the measured BD values. Once, the appropriate PTF was chosen, the `estimateBD` function is applied in the dataframe `dat` that was created at the end of the quality check. Here, new bd values are estimated for the rows in which the column 'bd' has missing values. Finally, a plot is generated to visualize the gap-filled bulk density values.

```
## 5.4 Estimate BD for the missing horizons ------------------------------------
dat$bd[is.na(dat$bd)] <-
  estimateBD(dat[is.na(dat$bd),]$soc, method="Honeyset_Ratkowsky1989")

# Explore the results
summary(dat$bd)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.4192  1.2624  1.5797  1.4773  1.7008  1.7670     356
```

```
g <- BD_test %>%
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density() +
  xlim(c(0,2.5))
g + geom_density(data = dat, aes(x=bd, color = "Predicted +\n observed"))
```

```
## Warning: Removed 356 rows containing non-finite values (`stat_density()`).
```



6.4 Check for outliers

Unrealistically high or low values can have considerable impact on the statistical analysis and thus it is key to identify and carefully check those values in order to get valid results and eliminate potential bias. Again, the summary() function is apt to show general descriptive statistics such as maxima or minima. Based on this assessment, more detailed views of the suspicious values can be obtained by filtering values above or below a certain threshold as done in the code below for soil organic carbon (SOC) values above 10 percent. If such values don't belong to soil types that would justify such exceptionally high SOC values, e.g. organic soils (Histosols), these rows can be removed based on the profile ID. The same process should be repeated for all soil properties. Such evaluation can also be conducted visually for several properties at the same time using the `tidyverse` and `ggplot` package that allows to plot boxplots for several soil properties at the same time. To get more information on tidyverse, please follow this link: https://r4ds.had.co.nz/. For a comprehensive overview of the functionalities of

ggplot, a more sophisticated way of plotting, this book provides a good overview: http://www.cookbook-r.com/Graphs/.

```
## 5.5 - Explore outliers ----------------------------------------------------
# Outliers should be carefully explored and compared with literature values.
# Only if it is clear that outliers represent impossible or highly unlikely
# values, they should be removed as errors.
#
# Carbon content higher than 15% is only typical for organic soil (histosols)
# We will remove all atypically high SOC as outliers
summary(dat$soc)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.020   0.250   0.720   1.451   2.340  19.000     356
```

```
na.omit(dat$ProfID[dat$soc > 10])
```

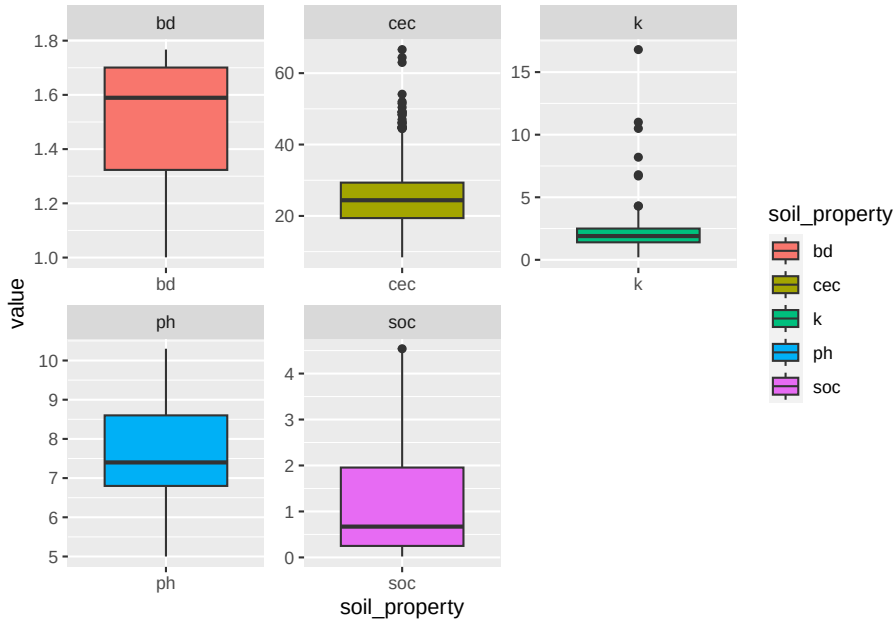
```
## [1] 6915 7726
## attr(,"na.action")
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
## [109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 124 125 126 127
## [127] 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
## [145] 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
## [163] 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
## [181] 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
## [199] 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217
## [217] 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235
## [235] 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253
## [253] 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
## [271] 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
## [289] 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307
## [307] 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325
## [325] 326 327 328 329 330 331 332 333 334 335 336 337 339 340 341 342 343 344
```

```
## [343] 345 346 347 348 349 350 351 352 353 354 355 356 357 358
## attr(,"class")
## [1] "omit"
```

```
dat$ProfID[dat$soc > 10][!is.na(dat$ProfID[dat$soc > 10])]
```

```
## [1] 6915 7726
```

```
dat <- dat[dat$ProfID != 6915,]
dat <- dat[dat$ProfID != 7726,]

dat<- dat[!(dat$ProfID %in% dat$ProfID[dat$soc > 10][!is.na(dat$ProfID[dat$soc >

# Explore bulk density data, identify outliers
# remove layers with Bulk Density < 1 g/cm^3
low_bd_profiles <- na.omit(dat$ProfID[dat$bd<1])
dat <- dat[!(dat$ProfID %in% low_bd_profiles),]

# Explore data, identify outliers
x <- pivot_longer(dat, cols = ph:cec, values_to = "value",
                  names_to = "soil_property")
x <- na.omit(x)
ggplot(x, aes(x = soil_property, y = value, fill = soil_property)) +
  geom_boxplot() +
  facet_wrap(~soil_property, scales = "free")
```



6.5 Harmonise soil layer depths

The last step towards a soil data table that can be used for mapping, is to harmonize the soil depth layers to 0-30 cm (or 30-60, or 60-100 cm respectively). This is necessary since we want to produce maps that cover exactly those depths and do not differ across soil profile locations. Thus, the relevant columns are selected from the dataframe, target soil properties, and upper and lower limit of the harmonised soil layer are specified (in depths).

In the following a new dataframe 'd' is created in which the standard depth layers are stored and named. The code below shows a for loop that calculates the values for the standard depth for each target soil property automatically using the ea_spline function of the 'ithir' package.

```
# 6 - Harmonize soil layers ============================================
source("Digital-Soil-Mapping/03-Scripts/spline_functions.R")
## 6.1 - Set target soil properties and depths ---------------------------
```

```
names(dat)
dat <- select(dat, ProfID, HorID, x, y, top, bottom, ph, k, soc, bd, cec)

target <- c("ph", "k", "soc",  "bd", "cec")
depths <- c(0,30)

## 6.2 - Create standard layers ----------------------------------------------
splines <- apply_mpspline_all(df = dat, properties = target, depth_range = depths
summary(splines)

# merge splines with x and y
d <- unique(select(dat, ProfID, x, y))
d <- left_join(d, splines)
```

6.6 Harmonise units

Units are of paramount importance to deliver a high-quality map product. Therefore, special attention needs to be paid to a correct conversion/harmonisation of units particularly if different spreadsheets are combined. The mandatory soil properties need to be delivered in the following units:

```
# 7 - Harmonise units ========================================================
#Harmonise units if different from target units
# Mandatory Soil Properties and corresponding units:
# Total N - ppm
# Available P - ppm
# Available K - ppm
# Cation exchange capacity cmolc/kg
# pH
# SOC - %
# Bulk density g/cm3
# Soil fractions (clay, silt and sand) -
```

In the following, the available Potassium measurements from the soil profile data is converted from $cmol_c/kg$ to ppm. In addition, total N is converted from percent to ppm and the soil texture class values from g/kg to percent.

```
# Units soil profile data (dataframe d)
#
head(d) # pH; K cmolc/kg; SOC %; BD g/cm3; CEC  cmolc/kg
```

```
## # A tibble: 6 x 8
##   ProfID     x     y ph_0_30 k_0_30 soc_0_30 bd_0_30 cec_0_30
##    <dbl> <dbl> <dbl>   <dbl>  <dbl>    <dbl>   <dbl>    <dbl>
## 1    154 -58.7 -38.2    6.03   2.10    3.07     1.17     24.8
## 2    197 -60.5 -38.4    5.95   2.47    2.93     1.19     24.2
## 3    262 -58.9 -38.4    6.63   2.20    1.67     1.38     18.2
## 4   2702 -58.0 -37.8    7.09   2.21    3.78     1.08     28.4
## 5   2706 -57.9 -38.0    6.00   2.28    3.51     1.12     23.1
## 6   2709 -60.5 -36.7   10.0    2.95    0.884    1.55     NA
```

```
# K => convert cmolc/kg to ppm (K *10 * 39.096)
d$k_0_30 <- d$k_0_30*10 * 39.096

head(chem)# P ppm; N %; K ppm
```

```
## # A tibble: 6 x 6
##   LabID     x     y p_bray     k    tn
##   <chr> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 51    -61.5 -37.4   20.4  852. 0.225
## 2 60    -57.8 -37.9   10.5  770. 0.301
## 3 64    -58.9 -38.5   15.9  992. 0.266
## 4 67    -60.3 -38.5   20.8  740. 0.179
## 5 68    -60.4 -38.5   13.5  725. 0.168
## 6 69    -60.4 -38.5   46.2  699. 0.129
```

```
# N => convert % to ppm (N * 10000)
chem$tn <-chem$tn*10000

head(phys)# clay, sand, silt g/kg
```

```
## # A tibble: 6 x 6
##   ProfID     x     y clay_0_30 sand_0_30 silt_0_30
```

```
##    <dbl> <dbl> <dbl>     <dbl>     <dbl>     <dbl>
## 1    154 -58.7 -38.2      260.       410      330.
## 2    197 -60.5 -38.4      251.       400      349.
## 3    262 -58.9 -38.4      213.       480      307.
## 4   2702 -58.0 -37.8      260.       430      310.
## 5   2706 -57.9 -38.0      265.       400      335.
## 6   2709 -60.5 -36.7      324.       310      366.
```

```
# convert g/kg to % (/10)
phys$clay_0_30 <-phys$clay_0_30/10
phys$sand_0_30  <-phys$sand_0_30 /10
phys$silt_0_30 <-phys$silt_0_30/10
```

Finally, the different spreadsheets are merged into one single dataframe. For that, it is important to have matching column names in the dataframes that are to be merged.

```
# Add chemical and physical properties from additional datasets ================


# Rename columns to match the main data set
names(d)
```

```
## [1] "ProfID"   "x"        "y"        "ph_0_30"  "k_0_30"   "soc_0_30" "bd_0_30
## [8] "cec_0_30"
```

```
names(chem)[1] <- 'ProfID'
names(chem)[4] <- 'p_0_30'
names(chem)[5] <- 'k_0_30'
names(chem)[6] <- 'n_0_30'


#The chem dataframe comes from and independent dataset we need to create new uniq
#Create unique ProfID
chem$ProfID <- seq(max(d$ProfID)+1,max(d$ProfID)+1+nrow(chem)-1)

# Add the new data as new rows using dplyr we can add empty rows
```

```
# automatically for the not measured properties in the chem dataset
d <- bind_rows(d, chem)

#The phys dataframe with the texture instead shares the same ProfIDs (we can dire
d <- left_join(d, phys, by=c('ProfID', 'x', 'y'))
```

6.7 Save the results

Before finalising the soil data preparation, it is recommendable to check again visually if the calculations were conducted correctly. Again, the combination of tidyverse and ggplot functions provides high efficiency and versatility to visualise figures with the desired soil properties. At last, the write_csv() function is used to save the dataframe as a .csv file in the Outputs folder (02-Outputs). With this, the soil data preparation is finalised.

```
# 8 - Plot  and save results ======================================================
names(d)
x <- pivot_longer(d, cols = ph_0_30:silt_0_30, values_to = "value",
                  names_sep = "_",
                  names_to = c("soil_property", "top", "bottom"))
x <- mutate(x, depth = paste(top, "-" , bottom))
#x <- na.omit(x)
ggplot(x, aes(x = depth, y = value, fill = soil_property)) +
  geom_boxplot() +
  facet_wrap(~soil_property, scales = "free")


ggplotly(ggplot(x, aes(x = depth, y = value, fill = soil_property)) +
           geom_boxplot() +
           facet_wrap(~soil_property, scales = "free"))

# save the data
write_csv(d, "02-Outputs/harmonized_soil_data.csv")
```

Chapter 7

Step 2: download environmental covariates

7.1 Environmental covariates

The SCORPAN equation (Eq. (4.1)) refers to the soil-forming factors that determine the spatial variation of soils. However, these factors cannot be measured directly. Instead, proxies of these soil forming factors are used. One essential characteristic of the environmental covariates is that they are spatially explicit, covering the whole study area. The following Table 7.1 lists all the environmental covariates that can be implemented under the present DSM framework. Apart from the environmental covariates mentioned in Table 7.1, other types of maps could also be included, such as Global Surface Water Mapping Layers and Water Soil Erosion from the Joint Research Centre (JRC). At national level there may be very significant covariates that could complement or replace the covariates of Table 7.1. Thus, the selection of suitable covariate layers needs to be assessed with common sense and applying expert knowledge.

Table 7.1: List of environmental covariates.

Description	Code	Resoluti
Temperature		
Mean air temperature (annual)	bio1	1000
Mean daily temperature of warmest month	bio5	1000
Mean daily temperature of coldest month	bio6	1000
Precipitation		
Mean annual precipitation	bio12	1000
Mean precipitation of wettest month	bio13	1000
Mean precipitation of driest month	bio14	1000
Mean monthly precipitation of wettest quarter	bio16	1000
Mean monthly precipitation of driest quarter	bio17	1000
Potential evapotranspiration (PET)		
Mean monthly PET	pet_penman_mean	1000
Minimum monthly PET	pet_penman_min	1000
Range monthly PET	pet_penman_range	1000
Maximum monthly PET	pet_penman_max	1000
Wind		
Minimum monthly wind speed	sfcWind_min	1000
Maximum monthly wind speed	sfcWind_max	1000
Range monthly wind speed	sfcWind_range	1000
Growing season		
Number of days with mean daily air temperature above 10 degrees Celsius	ngd10	1000
Vegetation indices (NDVI) (MOD13Q1)		
Mean March-May from 2000-2022	ndvi_030405_mean	250
Mean June-August from 2000-2022	ndvi_060708_mean	250
Mean September-November from 2000-2022	ndvi_091011_mean	250
Mean December-February from 2000-2022	ndvi_120102_mean	250
Standard deviation March-May (2000-2022)	ndvi_030405_sd	250
Standard deviation June-August (2000-2022)	ndvi_060708_sd	250
Standard deviation Sept.-Nov. (2000-2022)	ndvi_091011_sd	250
Standard deviation Dec.-Feb. (2000-2022)	ndvi_120102_sd	250
Fraction of photosynthetically active radiation (FPAR) (MOD15A2H)		
Mean March-May from 2000-2022	fpar_030405_mean	500
Mean June-August from 2000-2022	fpar_060708_mean	500
Mean September-November from 2000-2022	fpar_091011_mean	500
Mean December-February from 2000-2022	fpar_120102_mean	500
Standard deviation March-May (2000-2022)	fpar_030405_sd	500
Standard deviation June-August (2000-2022)	fpar_060708_sd	500
Standard deviation Sept.-Nov. (2000-2022)	fpar_091011_sd	500
Standard deviation Dec.-Feb. (2000-2022)	fpar_120102_sd	500
Land surface temperature day (LSTD) (MOD11A2)		
Mean March-May from 2000-2022	lstd_030405_mean	1000
Mean June-August from 2000-2022	lstd_060708_mean	1000
Mean September-November from 2000-2022	lstd_091011_mean	1000

7.2 Download covariatesand cropland mask with Google Earth Engine (GEE)

The following are the steps to access and download the environmental covariates and cropland mask. The GSP has streamlined the process of downloading environmental covariates by reducing the need to clip and download layers from GEE. This chapter aims to guide you on how to download environmental covariates using a GEE script.

You can find the JavaScript code in the material provided in this technical manual under 03-scripts/3.0.Download_Covariates_&_Mask.txt. To use the code, simply copy and paste the text in the GEE console as shown in the figure below.

If not done already, it is necessary to specify the working directory and a file path directory to the output folder where the clipped covariate layers are going to be saved. In case users want to use their own shapefile of the AOI, it is necessary to specify the file path to load it into our **R** session later. Alternatively, the shapefile of the AOI can be clipped from the official UN map shapefile that is available in the "Digital-Soil-Mapping-GSP-FAO" based on the 3-digit ISO code (ISO3CD column in the attribute table). The process to do this will be explained in a few steps. Finally, it is also necessary to specify the resolution to 250 x 250 m for the covariate layers and set the CRS to WGS84 (equals EPSG code 4326). Note that the target resolution of the GSNmap is at 250 m, which can be considered a moderate resolution for a global layer. However, those countries that require a higher resolution are free to develop higher resolution maps and aggregate the resulting maps to the target resolution of GSNmap for submission.

The following text explains the structure of the script that will be executed in GEE, and which parts of the script need to be modified to extract the covariates from the area of interest (AOI).

7.2.1 Assets

In GEE, an "asset" refers to any data or code that has been uploaded and stored in GEE's cloud-based servers. Assets can include remote sensing data, vector data, and even scripts or functions.

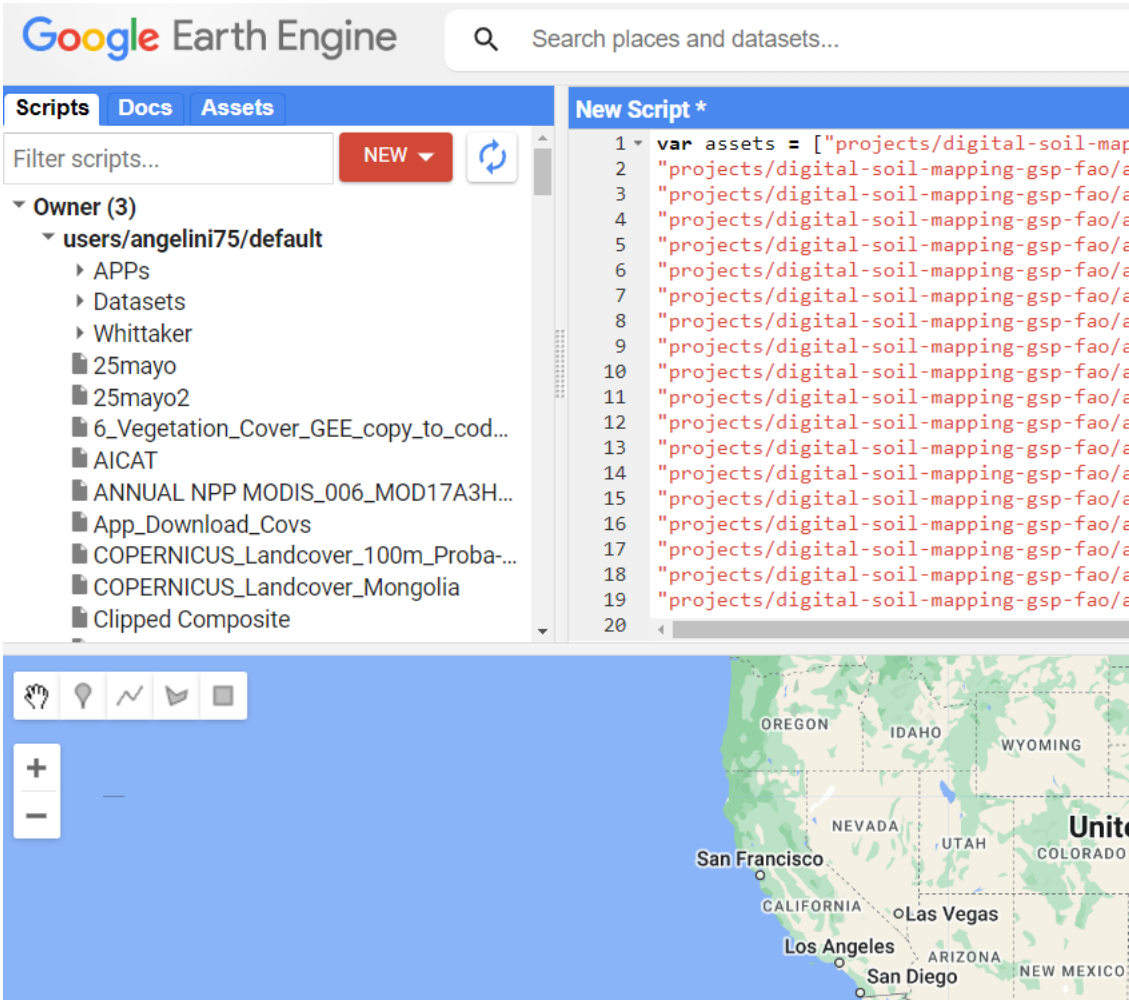


Figure 7.1: Copy and paste script in the code editor.

The following code reads the assets that have been created by the GSP in GEE. This means that the code accesses the data and scripts that GSP has uploaded to GEE, and uses them to perform specific tasks or analyses. By leveraging GEE's cloud-based infrastructure and GSP's assets, users can easily access and analyze large amounts of data without the need for local storage or processing power.

```
#Empty environment and cache
var assets = ["projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio1",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio12",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio13",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio14",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio16",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio17",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio5",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio6",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/ngd10",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_max",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_mean",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_min",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_range",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_max",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_mean",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_range",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_030405_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_030405_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_060708_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_060708_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_091011_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_091011_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_120102_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_120102_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_030405_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_030405_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_060708_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_060708_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_091011_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_091011_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_120102_mean",
```

```
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_120102_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_030405_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_030405_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_060708_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_060708_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_091011_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_091011_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_120102_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_120102_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_030405_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_030405_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_060708_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_060708_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_091011_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_091011_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_120102_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_120102_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/snow_cover",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/swir_060708_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/crops",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/flooded_vegetation",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/grass",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/shrub_and_scrub",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/trees",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_curvature_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_downslopecurvature_
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_dvm2_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_dvm_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_elevation_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_mrn_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_neg_openness_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_pos_openness_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_slope_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_tpi_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_twi_500m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_upslopecurvature_25
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_vbf_250m"];
```

7.2.2 Define the region of interest (ROI)

This script in GEE loads borders of a specific country or a user-defined shapefile into the workspace. It does this by creating a region of interest (ROI) based on the country borders. The script first specifies a list of countries to be included in the ROI and then loads the corresponding geometries from the 'USDOS/LSIB_SIMPLE/2017' feature collection in the case of the LSIB 2017 dataset. In the case of a user-defined shapefile, the script uploads the borders of the ROI as an asset and replaces 'your_shapefile' with the path to the uploaded shapefile. Finally, the region variable is assigned the geometry of the ROI, which can be used to clip and process data within the specified boundary. You must change either the name of the country or the shape file (as an asset) to download the covariates for your specific ROI.

```
// Load borders

/// Using LSIB 2017 (replace the countries that you want to download)
var country_list = ['Italy'];
var aoi = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
  .filter(ee.Filter.inList('country_na', country_list));
var region = aoi.geometry();

/// Using a shapefile
/// 1. Upload the borders of your countries as an asset
/// 2. Replace 'your_shapefile' with the path to your shapefile
// var shapefile = ee.FeatureCollection('users/your_username/your_shapefile');
// var region = shapefile.geometry();
```

7.2.3 Load and clip the covariates

This script loads an asset collection into an Earth Engine (EE) image collection and clips each image in the collection to a specific region of interest (ROI).

First, the ee.ImageCollection function is used to load the assets into an EE image collection. The assets variable is expected to be a list of asset IDs.

Next, the map function is applied to the image collection to clip each image to the specified ROI. The clip function clips the image to the given ROI, and toFloat() converts the data type of the clipped image to floating-point values.

The result of this operation is a new EE image collection called clippedCollection, where each image has been clipped to the specified ROI.

```
// Load assets as ImageCollection
var assetsCollection = ee.ImageCollection(assets);

// Clip each image in the collection to the region of interest
var clippedCollection = assetsCollection.map(function(img){
  return img.clip(region).toFloat();
});
```

7.2.4 Clean holes in FPAR layers

The Fraction of Photosynthetically Active Radiation (FPAR) MODIS product represents the fraction of incident photosynthetically active radiation (PAR) that is absorbed by vegetation. This product is calculated from satellite-based observations of surface reflectance, and it is commonly used to estimate vegetation growth and productivity.

In some areas, the FPAR MODIS product contains no data values in areas where the vegatation is scarce or absent. To avoin transferring these holes to the digital soil maps, we covert no data values to zeroes. The rest of the script in this section is to reclip the rasters, stack them in a single object and rename them.

```
// Function to replace masked values with zeroes for fpar bands
function replaceMaskedFpar(img) {
  var allBands = img.bandNames();
  var fparBands = allBands.filter(ee.Filter.stringStartsWith('item', 'fpar'));
  var nonFparBands = allBands.removeAll(fparBands);

  var fparImg = img.select(fparBands).unmask(0);
  var nonFparImg = img.select(nonFparBands);

  // If there are no fpar bands, return the original image
  var result = ee.Algorithms.If(fparBands.length().eq(0),
                                img,
                                nonFparImg.addBands(fparImg));
```

```
  return ee.Image(result);
}

// Clip each image in the collection to the region of interest and replace masked
var clippedCollection = assetsCollection.map(function(img){
  var clippedImg = img.clip(region).toFloat();
  return replaceMaskedFpar(clippedImg);
});

// Stack the layers and maintain the layer names in the final file
var stacked = clippedCollection.toBands();

// Get the list of asset names
var assetNames = ee.List(assets).map(function(asset) {
  return ee.String(asset).split('/').get(-1);
});

// Rename the bands with asset names
var renamed = stacked.rename(assetNames);
print(renamed, 'Covariates to be exported')
```

7.2.5 Visualize and export the covariates

This script has two main parts: visualizing the result and exporting the stacked image to Google Drive.

In the first part, the script sets a visualization parameter (visParams) to define the visualization properties of the stacked image. Specifically, the script specifies that the visualization should use the 'bio1' band and a color palette with four colors to represent the range of values in the band. The min and max values are also set to control the range of values that are displayed.

Next, the script adds the renamed image (renamed) to the map and applies the visualization parameters defined in visParams. The Map.centerObject() function centers the map on the renamed image, and the Map.addLayer() function adds the image layer to the map with the specified name ('Covariates').

In the second part of the script, the Export.image.toDrive() function is used to

export the stacked image to Google Drive. This function exports the renamed image as a GeoTIFF file with the description 'covariates', which is saved in the 'GEE' folder in the user's Google Drive (you have to either create the folder in GDrive, or indicate a different target folder). The scale parameter specifies the spatial resolution of the exported image, while the maxPixels parameter sets the maximum number of pixels that can be exported. Finally, the region parameter specifies the geographic extent of the exported image, which is set to the region variable defined earlier in the script.

```
// Visualize the result
// Set a visualization parameter (you can adjust the colors as desired)
var visParams = {
  bands: 'bio1',
  min: 19248,
  max: 46139,
  palette: ['blue', 'green', 'yellow', 'red']
};

// Add the layer to the map
Map.centerObject(renamed, 6)
Map.addLayer(renamed, visParams, 'Covariates');


// Export the stacked image to Google Drive
Export.image.toDrive({
  image: renamed,
  description: 'covariates',
  folder: 'GEE',
  scale: 250,
  maxPixels: 1e13,
  region: region
});
```

7.2.6 Load and clip the Copernicus land cover map and

This script loads an image collection of global land cover, selects the 'discrete_classification' band, and clips the image to a specified region. It then sets the CRS and spatial resolution of the output image, and applies resampling

to change the spatial resolution to the desired value. The resulting image is stored in the variable image1.

```
/* Create mask for croplands ----------------------------*/

// Load the Copernicus Global Land Service image collection
var imageCollection = ee.Image("COPERNICUS/Landcover/100m/Proba-V-C3/Global/2019"
  .select("discrete_classification")
  .clip(region)

var crs = 'EPSG:4326'; // WGS84
var res = 250; // Resolution in decimal degrees

// Default resampling is nearest neighbor
var image1 = imageCollection.resample()
  .reproject({
    crs: crs, // Add your desired CRS here
    scale: res // Add your desired scale here
  });
```

7.2.7 Reclassify the land cover map

This script reclassifies the land cover classes of the image1 using the remap function, which replaces the values in inList with the corresponding values in outList. We only keep class 40 which refer to Cultivated and managed vegetation / agriculture. The resulting image is then converted to a double data type, clipped to a specified region, and stored in the variable FAO_lu. The script then converts all 0 values in FAO_lu to NA values using the updateMask function and stores the resulting masked image in FAO_lu. The intermediate results are printed using the print function.

```
// Reclassify the land cover classes
var inList = [0, 20, 30, 40, 50, 60, 70, 80, 90, 100, 111, 112, 113, 114, 115, 11
              121, 122, 123, 124, 125, 126, 200];
var outList = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

var FAO_lu = image1.remap(inList, outList)
```

```
  .toDouble()
  .clip(region);

// print(FAO_lu)

// Convert 0 to NA
var mask = FAO_lu.neq(0);
print(mask)
FAO_lu = FAO_lu.updateMask(mask);

print(FAO_lu, "Mask")
```

7.2.8 Visualization and exporting mask

The code sets up visualization parameters for the reclassified land cover image and adds it as a layer to the map using the Map.addLayer function. The resulting image is then exported as a raster to Google Drive using the Export.image.toDrive function with specified parameters such as the folder to save the image, the desired scale and CRS, the region of interest, and the maximum number of pixels for export if needed. The resulting image is a binary mask where 1 represents the forest class and 0 represents all other classes.

```
var visParams = {
  bands: 'remapped',
  min: 0,
  max: 1,
  palette: ['green', 'yellow']
};

// Add the layer to the map
Map.addLayer(FAO_lu,visParams ,'Mask');

// Export the land cover image as a raster to Google Drive
Export.image.toDrive({
  image: FAO_lu,
  folder: 'GEE',
  description: 'mask',
```

```
  scale: res, // Add your desired scale here
  region: region,
  crs: crs, // Add your desired CRS here
  maxPixels: 1e13 // Add a maximum number of pixels for export if needed
});
```

7.3 Run and export in GEE

To execute a script in GEE, you can run it by clicking the "Run" button in the upper right-hand corner of the code editor. The "Run" button in GEE executes the script and any tasks specified in the script, such as exporting files to Google Drive. The status of the task can be monitored in the "Tasks" tab.

To run a task for exporting files to Google Drive in GEE, the Export.image.toDrive() function exports the image Google Drive. To start the task, you need to RUN the task, and it will be added to the GEE task list. You can monitor its progress and download the exported file once the task is complete.

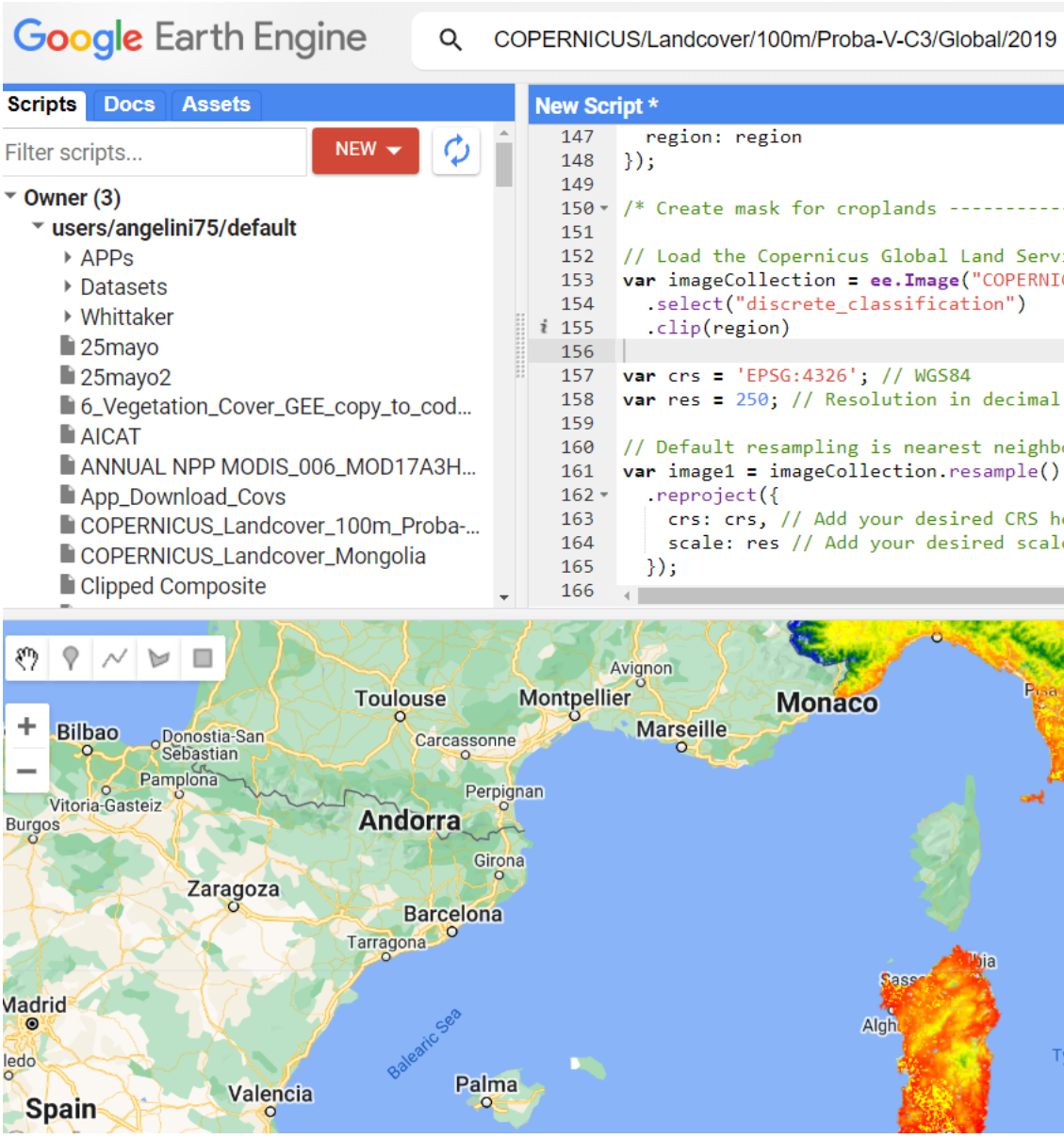


Figure 7.2: Run button in code editor and RUN task in Tasks bar.

Chapter 8

Step 3: Mapping continuous soil properties

In this chapter, the cleaned soil data and the previously downloaded covariate layers are used to generate soil property maps using DSM techniques. These consist of merging soil and environmental covariates data, selecting the covariates, calibrating the machine learning model, assessing the uncertainty, predicting the soil properties and finally export the maps.

8.1 Mapping Soil Properties with Random Forest

Random Forest is a popular machine learning algorithm used in Digital Soil Mapping (DSM) for predicting the distribution of continuous soil properties. It leverages an ensemble of decision trees to improve the accuracy of predictions for the expected value of the target soil parameters.

In Random Forest, multiple individual decision trees are created, each trained on a random subset of the available data. These trees make independent predictions, and the final prediction is obtained by aggregating the predictions of all the trees. This ensemble approach helps to reduce the risk of overfitting and improves the model's generalization ability. Furthermore, Random Forest introduces an additional level of randomness by selecting different subsets of

covariates (features) at each node of the trees. This feature sampling increases the independence between trees and helps mitigate potential collinearity issues in the individual regression tree models.

By combining the predictions of multiple trees and incorporating randomness in feature selection, Random Forest provides robust and reliable predictions for soil properties in DSM applications.

The main limitation of Random Forest is that it focuses primarily on predicting the mean value of the target variable thus it does not provide direct information about the variability or distribution of the target variable's predictions. This limits the possibility of assessment of the uncertainty of the predictions.

Quantile regression forests (QRF, Meinshausen (2006)) are a generalisation of the random forest models, capable of not only predicting the conditional mean, but also the conditional probability density function. This provides a more comprehensive understanding of the full conditional distribution of the predictions, capturing the uncertainty and dispersion associated with different quantiles or levels of the target variable. This feature allows one to estimate the standard deviation of the prediction, as well as the likelihood of the target variable falling below a given threshold. By considering the dispersion of the predictions, QRF enables us to assess the model uncertainty and make more informed decisions. In a context where a minimum level of a soil nutrient concentration may be decisive for improving the crop yield, this feature can play an important role for the GSNmap initiative. Model calibration will be implemented using the caret package (Kuhn, 2022). While we suggest to use QRF, caret provides a large set of models https://topepo.github.io/caret/available- models.html#) that might perform better in specific cases. In this regard, it is up to the user to implement a different model, ensuring the product specifications (Section Product Specifications).

We demonstrate the implementation of Quantile Regression Forest (QRF) for modeling and mapping soil properties and its associated uncertainty. The implementation presented here utilizes the Boruta, ranger, caret, and terra R packages.

8.2 Getting prepared to map

To begin, we open *RStudio* and empty our global environment. Then, we set the working directory and assign the file path to our AOI shapefile to an R object. The target soil property that is going to be mapped in this exercise is Potassium denoted as 'k' in the soil data table. Next, an R function that was built by the GSP is loaded from the training material folder. Finally, the packages that are going to be needed for mapping are called.

```
#-------------------------------------------------------------------------------
#
# Quantile Regression Forest
# Soil Property Mapping
#
# GSP-Secretariat
# Contributors: Isabel Luotto (GSP-FAO)
#               Marcos E. Angelini (GSP-FAO)
#               Luis Rodríguez Lado (GSP-FAO)
#               Stephen Roecker (NRCS-USDA)
#-------------------------------------------------------------------------------

#Empty environment and cache
rm(list = ls())
gc()

# Content of this script ======================================================
# 0 - Set working directory, soil attribute, and packages
# 1 - Merge soil data with environmental covariates
# 2 - Covariate selection
# 3 - Model calibration
# 4 - Uncertainty assessment
# 5 - Prediction
# 6 - Export final maps
#-------------------------------------------------------------------------------


# 0 - Set working directory, soil attribute, and packages =====================
```

```
# Working directory
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd("..")

# Define country of interes throuhg 3-digit ISO code
ISO ='ISO'

# Load Area of interest (shp)
AOI <- '01-Data/AOI.shp'

# Terget soil attribute (Mandatory 10)
soilatt<- "soc_0_30"

# Function for Uncertainty Assessment
load(file = "03-Scripts/eval.RData")

#load packages
library(tidyverse)
library(caret)
library(terra)
library(Boruta)
library(ranger)
```

Since soil data and environmental covariates are stored in different files and formats, it is necessary to first merge them into one dataframe. For this purpose, the covariate raster file is loaded into **R** from the covariates folder. Secondly, the table with the cleaned and quality checked soil data is loaded and converted to a shapefile using the lat/long coordinates columns.

```
# 1 - Merge soil data with environmental covariates ============================

## 1.1 - Load covariates -------------------------------------------------------
covs <- rast("01-Data/covs/Covariates.tif") # match case of the file name
ncovs <- names(covs)

## 1.2 - Load the soil data (Script 2) -----------------------------------------
dat <- read_csv("02-Outputs/harmonized_soil_data.csv")
```

```
# Convert soil data into a spatial object (check https://epsg.io/6204)
dat <- vect(dat, geom=c("x", "y"), crs = crs(covs))
```

The shapefile can be reprojected to match the CRS of the covariates using the project function of the terra package.

```
# Reproject point coordinates to match coordinate system of covariates
dat <- terra::project(dat, covs)
names(dat)
```

Afterwards, the extract function can be used to extract the values of each covariate raster layer at the point location of each soil profile. This data is then merged in the dat dataframe. After checking the descriptive statistics of dat with the `summary()` command, the target soil attribute is selected together with the covariates. Finally, NA values (empty row values) are removed using the `na.omit()` function.

```
## 1.3 - Extract values from covariates to the soil points ----------------------
pv <- terra::extract(x = covs, y = dat, xy=F)
dat <- cbind(dat,pv)
dat <- as.data.frame(dat)

summary(dat)
```

8.3 Covariate selection

In a high-dimensional datasets, not all available features are equally informative for training a model. By detecting and subsetting the only relevant features for modeling, we can improve model's predictive performance, reduce its complexity, speeds up computations and enhance interpretability, allowing us to gain deeper insights into the underlying relationships within the data. Feature selection serves this purpose by identifying and retaining only those relevant features that are uncorrelated and non-redundant in the prediction of a target soil parameter, resulting in a more focused and effective model that is capable of making more accurate predictions. Feature selection was implemented through the Boruta package (Kursa and Rudnicki, 2010), a feature selection algorithm

that aims to identify the most relevant features in a dataset in both classification and regression problems. The algorithm is specifically designed for dealing with high-dimensional datasets and works by comparing the importance of each covariate against a randomized version of itself, known as shadow feature, that serve as a reference to determine whether the covariate has a higher importance than expected by chance. The algorithm goes through a series of iterations, where it progressively eliminates irrelevant features. In each iteration, it evaluates the importance of each feature based on the random forest model and compares it to the importance of the corresponding shadow feature. If a feature's importance is significantly higher than its shadow, it is considered relevant and retained. Otherwise, it is deemed uninformative and removed. The iterations continue until all features are either confirmed as relevant or identified as uninformative. The final outcome is a set of relevant features (covariates) that are statistically significant in terms of their importance for explaining the target variable, thus reducing data dimensionality for improved model performance and interpretability. The feature importance can be displayed in a graph showing different colors according to the importance of each feature in the model, with 'green', 'yellow', 'red' and 'blue' colors for 'confirmed', 'tentative', 'rejected' and 'shadow' features respectively (see 8.1). Only those marked in green color are retained as valuable predictors.

```
# 2 - Covariate selection with Boruta package ================================
# Wrapper feature selection algorithm
## 2.1 - Run the Boruta algorithm ------------------------------------------
fs_bor <- Boruta(y = d[,soilatt], x = d[-1], maxRuns = 100, doTrace = 1)

## 2.2 - Plot variable importance and selected features ----------------------
png(filename = paste0("02-Outputs/importance_",soilatt,".png"),
     width = 15, height = 25, units = "cm", res = 600)
par(las = 2, mar = c(4, 10, 4, 2) + 0.1)
Boruta:::plot.Boruta(fs_bor, horizontal = TRUE, ylab = "",
                     xlab = "Importance", cex.axis=0.60)
dev.off()
## 2.3 - Extract the selected feature variables ------------------------------
(fs_vars <- getSelectedAttributes(fs_bor, withTentative = TRUE))
```

The selected covariates can be visualised in Trellis displays. Finally, the optimal predictors are stored in a dedicated R object.

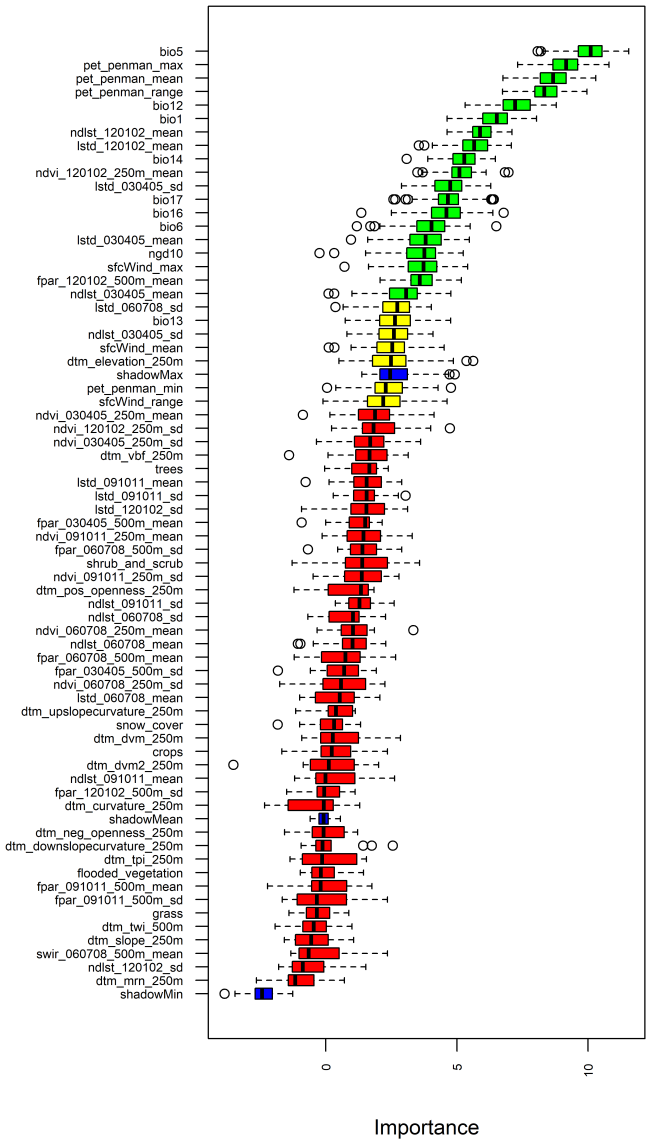


Figure 8.1: Covariates selecction for Total Nitrogen using Boruta algorithm.

8.4 Model calibration

The QRF model is calibrated using only the previous selection of covariates that actually have shown an effect on the target soil property. This is an important step to avoid overfitting of the model that can hamper a model's capacity to predict. Calibration is done by the 'train' function from the caret package, utilizing the implementation of the random forest algorithm provided by the ranger package. During the model training, several important hyperparameters need to be set. These hyperparameters include:

- 'mtry': This parameter determines the number of subset predictors considered for each split in the regression trees. It controls the level of randomness and feature diversity in the tree-building process.
- 'splitrule': This parameter specifies the criterion used to evaluate the quality of a potential split at each node of the tree. For regression problems, it is often set to "variance," which measures the reduction in variance of the target variable achieved by splitting at a particular node. In our implementation, we consider three possible criteria: "variance," "extratrees," and "maxstat." We select the criterion that produces the best calibrated model.
- 'min.node.size': This parameter represents the minimum number of observations (samples) required to create a terminal (leaf) node in a decision tree within the random forest algorithm. It ensures that each leaf contains a minimum number of observations to prevent overfitting and improve the generalization ability of the model. In regression problems, this number is typically set to 5.

Additionally, certain control parameters are necessary for training the model. These parameters are defined in the trainControl function and relate to the resampling method used to construct the bootstrap datasets. In our case, the resampling method is set to 'repeatedcv,' with 10-fold cross-validation and 10 repetitions.

As cross-validation plays a crucial role in the model calibration step, we explain the process in detail at this stage. Cross-validation is a widely used method in Digital Soil Mapping (DSM) to assess the overall accuracy of the resulting maps. It involves randomly dividing the input data into a training set and a

testing set. However, relying on a single testing dataset can introduce bias in the overall accuracy estimation. To mitigate this bias, we employ k-fold cross-validation. In this approach, the data is randomly partitioned into k parts, with one part used for testing and the remaining k-1 parts used for training the model. This process is repeated multiple times to enhance the robustness of parameter estimations. The final approach we adopt is known as repeated k-fold cross-validation, with k set to ten in this specific process. To help visualize the 10-fold cross-validation process, refer to Figure 8.2. Each row in the figure represents a step where the data is split into 10 subsets, with some samples marked as green balls representing the testing set, and others as yellow balls representing the training set. The figure illustrates how the data is divided into subsets in each step of the 10-fold process, and the blocks represent the repetition steps. By employing repeated k-fold cross-validation, we can obtain a comprehensive assessment of the model's performance, ensuring reliable and robust parameter estimations for the final model.

Repeated cross validation has been nicely implemented in the caret R package (Kuhn, 2022), along with several calibration methods. Here, we use the `Boruta()` function to specify the modalities of the cross-validation that contain the abovementioned settings. These settings are stored in an object called "fitControl". Next, the user has to specify a formula that will be used in a regression. In line with the purpose of mapping the target soil property, the formula has Potassium as target variable (dependent variable) and all covariates as independent or explanatory variables.

```
# 3 - QRF Model calibration with ranger ======================================
## 3.1 - Set training parameters -----------------------------------------------
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,          ## 10 -fold CV
                           repeats = 10,         ## repeated 10 times
                           savePredictions = TRUE)

## 3.2 - Tune hyperparameters --------------------------------------------------
mtry <- round(length(fs_vars)/3)
tuneGrid <-  expand.grid(
  mtry = abs(c(mtry-round(mtry/2),
               mtry-round(mtry/3),
               mtry,
               mtry+round(mtry/3),
```

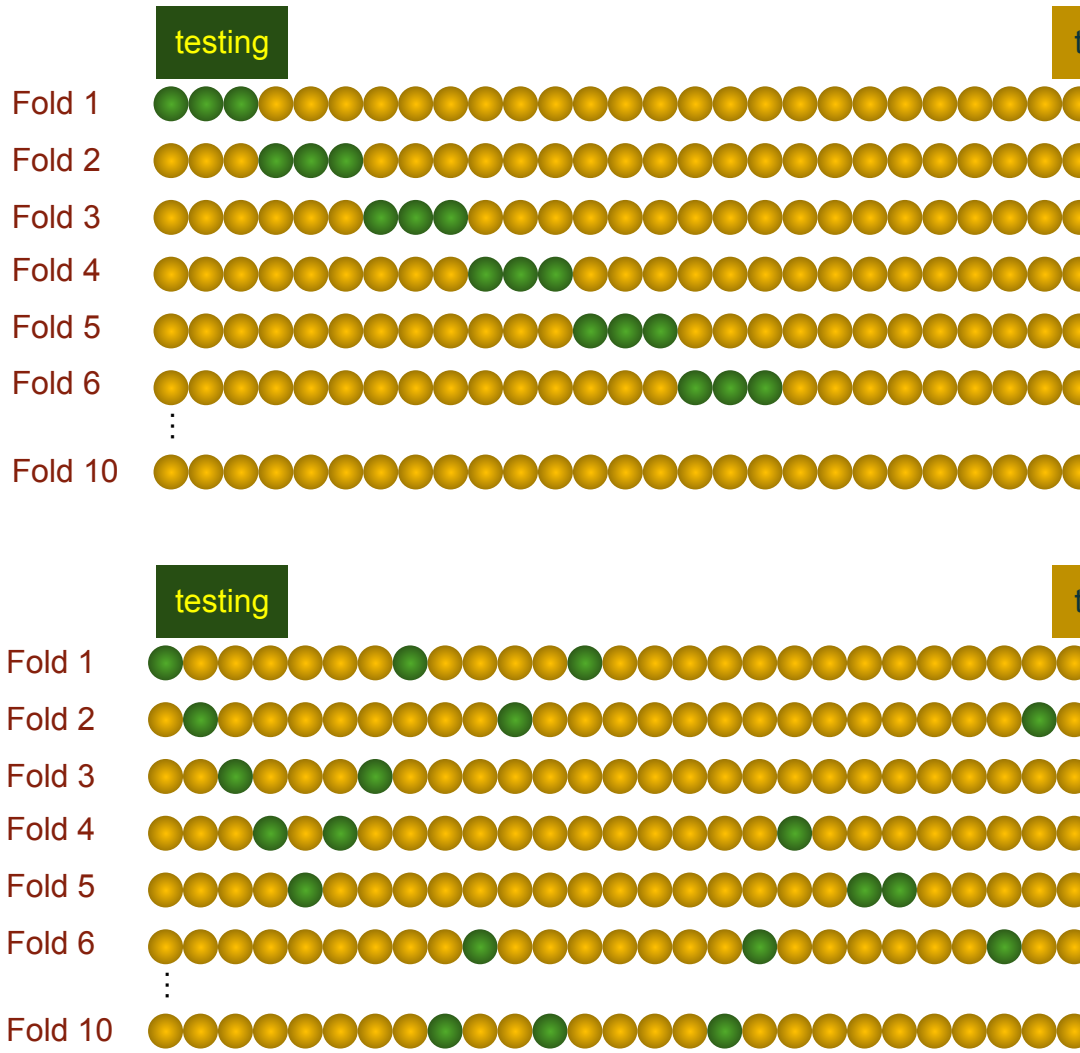


Figure 8.2: Schematic representation of the repeated cross-validation process.

```
                mtry+round(mtry/2))),
  min.node.size = 5,
  splitrule = c("variance", "extratrees", "maxstat")
)

## 3.3 - Calibrate the ranger model ----------------------------------------
print(soilatt)
print("training the model...")
model_rn <- caret::train(
  y = d[, soilatt], x = d[,fs_vars],
  method = "ranger",
  quantreg = TRUE,
  importance = "permutation",
  trControl = fitControl,
  verbose = TRUE,
  tuneGrid = tuneGrid
)
print(model_rn)
print(model_rn$bestTune)
```

The results have been stored in an R object called model_rn. To assess the contribution of each covariate on the model prediction. Finally, the model output is saved in the model folder within the Outputs folder - specifying the target soil properties.

8.5 Uncertainty assessment

Accuracy assessment is an essential step in digital soil mapping. One aspect of the accuracy assessment has been done in Step 7 by predicting the standard deviation of the prediction, which shows the spatial pattern of the uncertainty. Another aspect of the uncertainty is the estimation of the overall accuracy to measure the model performance. This will be measured using the model residuals generated by caret during the repeated cross validation step. The residuals produced by caret consist of tabular data with observed and predicted values of the target soil property. They can be used to estimate different accuracy statistics. Wadoux, Walvoort and Brus (2022) have reviewed and evaluated many of

them. While they concluded that there is not a single accuracy statistic that can explain all aspect of map quality, they recommended the following:

The average error indices all relate to the difference between observed (z) and predicted ($\hat{z}$) value of soil property S at the location i. The error ϵ is thus defined as:

$$\epsilon(S_i) = z(S_i) - \hat{z}(S_i) \tag{8.1}$$

The error indices that can be derived from this calculation inform about different aspects of prediction error and have the same unit as the target soil property. The mean prediction error (ME) estimates the prediction bias (see Eq. (8.2)). If the ME is negative it means that the predicted values are below the observed ones. Conversely, a positive ME indicates a bias of the model towards higher predictions.

$$ME = \frac{1}{N} \sum_{i=1}^{N} \epsilon(S_i) \tag{8.2}$$

Mean absolute error (MAE) and root-mean squared error (RMSE) estimate the magnitude of errors. The MAE takes the absolute value of the ME thus quantifies the overall magnitude of the prediction error (see Eq.(8.3)). The closer the MAE is to 0 the more accurate is the model prediction.

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\epsilon(S_i)| \tag{8.3}$$

Also, the RMSE provides a measure of the prediction error. Ideally, the RMSE approximates 0. Due to the squaring, larger absolute errors become more important (see Eq. (8.4)). Thus, high absolute errors may lead to a worse RMSE measure. Therefore, it is best to calculate all three error indices to get a comprehensive picture.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \epsilon(S_i)^2} \tag{8.4}$$

Besides the error indices, model quality can also be expressed by the coefficient of determination (R^2) which is the squared Pearson's product-moment correlation

coefficient (r) (see Eq. (8.5)). The R^2 takes values between 0 and 1. An R^2 of 1 indicates total correlation between predicted and observed values whereas 0 indicates no correlation. The R^2 can be biased by several factors and thus needs to be combined with other measures to yield a complete picture (Wadoux, Walvoort and Brus, 2022).

$$r^2 = \frac{\sum_{i=1}^{N}(z(S_i) - \overline{z})(\hat{z}(S_i) - \overline{z})}{\sqrt{\sum_{i=1}^{N}(z(S_i) - \overline{z})^2}\sqrt{\hat{z}(S_i) - \overline{\hat{z}})^2}} \tag{8.5}$$

The Pearson's product-moment correlation coefficient (r) can take values between -1 and 1 and thus indicate the direction of the correlation (see Eq. (8.6)).

$$r = \frac{\sum_{i=1}^{N}(z(S_i) - \overline{z})(\hat{z}(S_i) - \overline{z})}{\sqrt{\sum_{i=1}^{N}(z(S_i) - \overline{z})^2}\sqrt{\hat{z}(S_i) - \overline{\hat{z}})^2}} \tag{8.6}$$

The modelling efficiency coefficient (MEC) accounts for the proportion of variance that is explained by a model (Janssen and Heuberger, 1995). It is calculated as the ratio of the RMSE and the variance (squared standard deviation) (see Eq. (8.7)). In a perfect scenario, the MEC equals 1. If the MEC equals 0, it means that the model does not predict the values better than the mean of the observed values would. In addition to that, the MEC can also take negative values if the RMSE is greater than the variance. In consequence, negative MECs indicate that the model predicts the values worse than the mean of the observed values.

$$MEC = 1 - \frac{\sum_{i=1}^{N}(z(S_i) - \hat{z}(S_i))^2}{\sum_{i=1}^{N}(z(S_i) - \overline{z})^2} \tag{8.7}$$

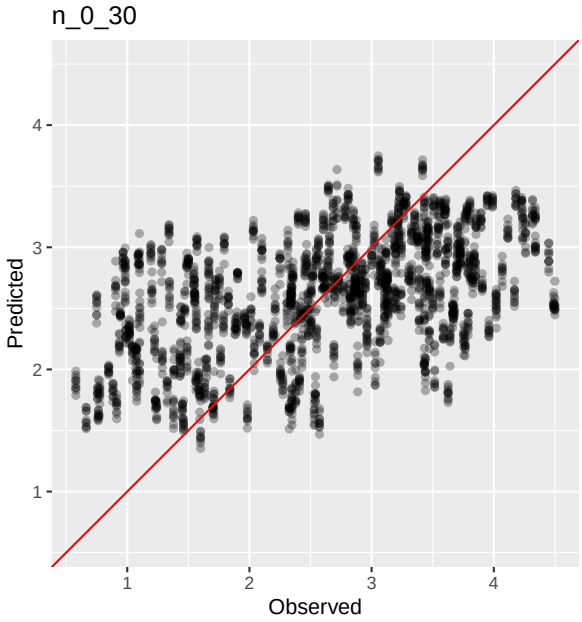
The R^2, RMSE, and the MEC are susceptible to bias through large error values. Thus, caution needs to be taken when interpreting the indices presented here for accuracy assessment.

Now, back to the mapping exercise: In practical terms, before calculating any of these indices, it is necessary to first extract observed and predicted values and then store them in two separate R objects. Next, both values are combined to a dataframe.

```
  # 4 - Accuracy assessment ============================================
  ## 4.1 - extract observed and predicted values ----------------------------
model_rn <- readRDS('Digital-Soil-Mapping/02-Outputs/models/ranger_model_soc_0_30

  o <- model_rn$pred %>%
    filter(mtry == model_rn$bestTune$mtry,
           splitrule==model_rn$bestTune$splitrule,
           min.node.size==model_rn$bestTune$min.node.size) %>%
    select(obs) %>% as.vector() %>% unlist()
  p <- model_rn$pred %>%
    filter(mtry == model_rn$bestTune$mtry,
           splitrule==model_rn$bestTune$splitrule,
           min.node.size==model_rn$bestTune$min.node.size) %>%
    select(pred) %>% as.vector() %>% unlist()
  df <- data.frame(o,p)
```

While solar diagrams (Wadoux, Walvoort and Brus, 2022) are desired, we propose to produce a scatterplot of the observed vs predicted values maintaining the same range and scale for the X and Y axes. The dataframe is used for this purpose to plot observed values on the x-axis and predicted values on the y-axis.

```
  ## 4.2 - Plot and save scatterplot ----------------------------------------
  (g1 <- ggplot(df, aes(x = o, y = p)) +
     geom_point(alpha = 0.3) +
     geom_abline(slope = 1, intercept = 0, color = "red")+
     ylim(c(min(o), max(o))) + theme(aspect.ratio=1)+
     labs(title = soilatt) +
     xlab("Observed") + ylab("Predicted"))
```




```
  # ggsave(g1, filename = paste0("02-Outputs/residuals_",soilatt,".png"), scale =
  #        units = "cm", width = 12, height = 12)
```

Additionally, it is necessary to calculate standard metrics of error estimation. The function eval() below returns values for the ME, RMSE, MAE, the squared pearson correlation coefficient, the concordance correlation coefficient, scale shift and location shift relative to scale.

```
## 4.2 - Print accuracy coeficients ----------------------------------------
# https://github.com/AlexandreWadoux/MapQualityEvaluation
print(eval(df$p,df$o))
```

```
##      ME  MAE RMSE    r   r2  MEC rhoC  Cb
## 1 0.01 0.68 0.84 0.51 0.26 0.26  0.4 0.8
```

Finally, note that accuracy assessment has been discussed in Wadoux *et al.* (2021), since the spatial distribution of soil samples might constrain the validity

Table 8.1: Accuracy statistics.

ME	MAE	RMSE	r	r2	MEC	rhoC	Cb
8.87	275.69	372.82	0.7	0.5	0.5	0.66	0.94

of the accuracy statistics. This is especially true in cases where the spatial distribution of observations is clustered. The authors recommended creating a kriging map of residuals before using them for assessing the map quality.

8.6 Predicting soil attributes

After calibrating the model, caret will select the best set of parameters and will fit the model using the whole dataset. Then, the final model can be used to predict the target soil properties. The process uses the model and the values of the covariates at target locations. This is generally done by using the same input covariates as a multilayer raster format, ensuring that the names of the layers are the same as the covariates in the calibration dataset. In this step we will predict the conditional mean and conditional standard deviation at each raster cell.

To prevent for potential computational power limitations, first the raster is split into so-called tiles that divide the whole area of interest in multiple rasters with a coarse resolution. In this case 25 tiles are produced (5 rows x 5 columns). The functions for tiling come from the terra package. The predictions can also be hadled at one time by setting the parameters 'nrows' and 'ncols' in the 't' object to 1.

```
# 5 - Prediction ==============================================================
# Generation of maps (prediction of soil attributes)
## 5.1 - Produce tiles ---------------------------------------------------------
# r <-covs[[1]]
# t <- rast(nrows = 5, ncols = 5, extent = ext(r), crs = crs(r))
# tile <- makeTiles(r, t,overwrite=TRUE,filename="02-Outputs/tiles/tiles.tif")
```

Next, a for loop is formulated to predict each soil attribute for each tile. The tiling significantly improves the computational speed of the prediction. For each

tile the mean and the standard deviation are stored in two separated objects that are then saved as raster files.

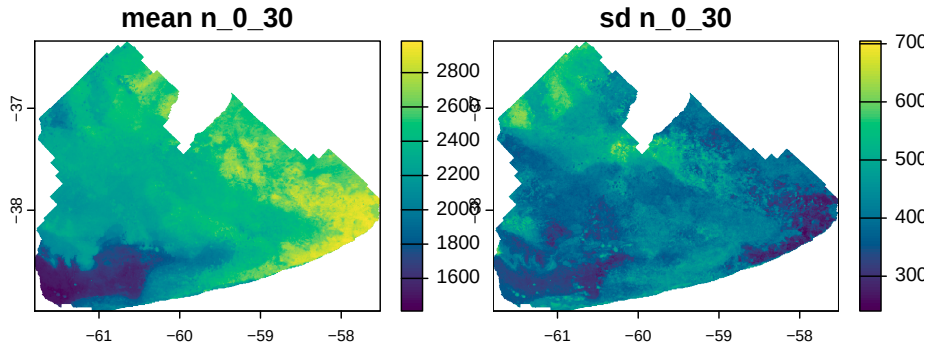
```
## 5.2 - Predict soil attributes per tiles -----------------------------------
# loop to predict soilatt on each tile

  for (j in seq_along(tile)) {
  gc()
  # read the tile
  t <- rast(tile[j])
  # crop the selected covariates with the tile j
  covst <- crop(covs[[fs_vars]], t)

  # create a function to extract the predited values from ranger::predict.range
  pfun <- \(...) { predict(...)$predictions |> t() }

  # predict conditional standard deviation
  terra::interpolate(covst,
                     model = model_rn$finalModel,
                     fun=pfun,
                     na.rm=TRUE,
                     type = "quantiles",
                     what=sd,
                     filename = paste0("02-Outputs/tiles/soilatt_tiles/",
                                       soilatt,"_tileSD_", j, ".tif"),
                     overwrite = TRUE)

  # predict conditional mean
  terra::interpolate(covst,
                     model = model_rn$finalModel,
                     fun=pfun,
                     na.rm=TRUE,
                     type = "quantiles",
                     what=mean,
                     filename = paste0("02-Outputs/tiles/soilatt_tiles/",
                                       soilatt,"_tile_", j, ".tif"),
                     overwrite = TRUE)

  print(paste("tile", j, "of", length(tile)))
```

```
}
```

As a result, 25 tiles for the predicted mean and 25 tiles for the predicted standard deviation were produced using the QRF model. The next step is to merge these tiles to produce a map of the predicted mean and one of the predicted standard deviation. For this, again for loops are employed that read all raster file tiles. These are then put together by the mosaic() function of the terra package. Finally, they are masked to the AOI and then can be visualised in a figure.

```
## 5.3 - Merge tiles both prediction and st.Dev --------------------------------
f_mean <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                     pattern = paste0(soilatt,"_tile_"), full.names = TRUE)
f_sd <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                   pattern =  paste0(soilatt,"_tileSD_"), full.names = TRUE)
r_mean_l <- list()
r_sd_l <- list()

for (g in 1:length(f_mean)){
  r <- rast(f_mean[g])
  r_mean_l[g] <-r
  rm(r)
}

for (g in 1:length(f_sd)){

  r <- rast(f_sd[g])
  r_sd_l[g] <-r
  rm(r)
}
r_mean <-sprc(r_mean_l)
r_sd <-sprc(r_sd_l)

pred_mean <- mosaic(r_mean)
pred_sd <- mosaic(r_sd)

aoi <- vect(AOI)
pred_mean <- mask(pred_mean,aoi)
pred_sd <- mask(pred_sd,aoi)
```

```
plot(c(pred_mean, pred_sd), main = paste(c("mean","sd"), soilatt),
     col = hcl.colors(100, "Viridis"))
```



The final step then consists of applying a cropland mask that is applied to the map and the uncertainty map since the soil data comes only from croplands and thus no assumption can be made to soil property values under different land covers. Additionally, a map is calculated to visualise the coefficient of variation (in Percent). The maps are then stored as raster files (GeoTiff/.tif) in the Outputs folder.

```
# 6 - Export final maps ================================================
## 6.1 - Mask croplands ------------------------------------------------
msk <- rast("01-Data/covs/mask.tif")
# plot(msk)
msk <- terra::project(msk, pred_mean)
pred_mean <- mask(pred_mean, msk)
```

```r
# plot(pred_mean)
pred_sd <- mask(pred_sd, msk)
# plot(pred_sd)
plot(pred_sd/pred_mean*100, main = paste("Coeficient of variation", soilatt),
     col = hcl.colors(100, "Viridis"))

## 6.2 - Save results ----------------------------------------------------------
writeRaster(pred_mean,
            paste0("02-Outputs/maps/",ISO,"_GSNmap_mean_",soilatt, ".tif"),
            overwrite=TRUE)
writeRaster(pred_sd,
            paste0("02-Outputs/maps/",ISO,"_GSNmap_sd_",soilatt, ".tif"),
            overwrite=TRUE)
```

Chapter 9

Reporting results

The GSNmap consists of a set of mandatory and optional maps that are to be generated as raster files (GeoTiff) at a resolution of 250 x 250 m and at the mandatory depth of 0-30 cm.

The Mandatory data products are:

- total N
- available P
- available K
- CEC
- soil pH
- clay, silt, and sand
- soil organic carbon concentration
- bulk density

Additionally, maps at deeper depths 30-60 cm and/or about micronutrients such as Ca, S, Mg, Fe, B, Cl, Mn, Zn, Cu, Mo, and Ni can be provided. All layers need to be submitted with the corresponding standard deviation layers.

An Rmarkdown script (with the extension .Rmd) is provided in the folder National Report. Script 5 can be used to translate the rmarkdown file into an automated report as a Word docx file.

```r
#_______________________________________________________________________________
#
# QA/QC
# Soil Property Mapping
#
# GSP-Secretariat
# Contact: Isabel.Luotto@fao.org
#          Marcos.Angelini@fao.org
#_______________________________________________________________________________

#Empty environment and cache
rm(list = ls())
gc()

# Content of this script ======================================================
# 0 - Setup and user-defined variables
# 1 -  User-defined variables
# 2 - Render the .Rmd file to generate an automated report as a docx
#_______________________________________________________________________________


# 0 - Initial Setup ===========================================================


#install.packages("rmarkdown")
#tinytex::install_tinytex()
# library(sf)
# library(ggplot2)
# library(tidyverse)
# library(terra)
library(knitr)
# library(tidyterra)
# library(patchwork)

# Working directory
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd("..")
```

```
# 1 -  User-defined variables ================================================

# Specify three-digit ISO code for your country
ISO <- 'AOI'

# Specify the properties you mapped (the code assumes harmonized naming)
# using the output data frame from script 2
dxy <- read.csv("02-Outputs/harmonized_soil_data.csv")

target_properties <- names(dxy)[ !(names(dxy)%in% c("ProfID", "x" ,"y"))]

#target_properties<-c('ph_0_30')

# Map background file
bckg<-vect('01-Data/AOI.shp')

#Adjust figure width and height of the map plot

figw <- 12
figh <-8


# Specify where you want your word document to be saved
output_file = paste0("National Report/Report_GSNmap_",ISO,".docx")

# 2 - Render the .Rmd file to generate an automated report as a docx ===========
path = 'National Report/National GSNmap Report.Rmd'
output_format = 'word_document'
rmarkdown::render(path, output_format, output_file)
```

The output report document is to be submitted along with the layers using a submission form provided by the GSP Secretariat.

Chapter 10

Way forward

This technical manual provided step-by-step guidance on how to generate nutrient maps by means of quantile regression forest models within a digital soil mapping framework. The array of maps produced belongs to the first implementation phase of the GSNmap initiative and provides urgently needed data on nutrient stocks and soil properties that govern nutrient availability. Policymakers will be able to use this data to derive conclusions on where to concentrate efforts to improve soil and land management to strengthen agrifood systems. The second phase of the GSNmap will make use of the first phase data products to derive nutrient budget maps. Therefore, additional datasets will be used for calculating input and output terms of nutrient stocks. The methodology is currently under development by the GSNmap working group. The technical documentation towards implementing the second phase will be made available in mid 2023.

10.1 Frequent asked questions and Troubleshooting answers

To be developed soon...

10.2 Issues in the GSNmap Technical Manual

Please, report any issue in the GSNmap Technical Manual in its issues GitHub page https://github.com/FAO-GSP/GSNmap-TM/issues.

10.3 Get help

- Check the issues GitHub page https://github.com/FAO-GSP/GSNmap-TM/issues
- Issues with R packages: search for solutions in https://stackoverflow.com/
- `caret` package https://topepo.github.io/caret/
- `terra` package https://rspatial.org/terra/pkg/1-introduction.html
- `tidyverse` package https://r4ds.had.co.nz/
- `sf` package https://r-spatial.github.io/sf/

If these links do not help you, contact us including the following text:

I am [FULL NAME], responsible for producing the GSNmap of [COUNTRY].

marcos.angelini@fao.org

Annex I: Compendium of R scripts

This chapter contains the complete list of R scripts to run the process of mapping soil nutrient and associated soil properties.

Script 0: Introduction to R

```
# Introduction to R

# 0. Playground ================================================================

# learn important keyboard shortcuts
# Ctrl + enter for running code
# tab after writing the first three characters of the function name
# F1 to access the help

# explore the use of <-, $, [], ==, !=, c(), :, data.frame(), list(), as.factor()

a <- 10:15
a[2]
a[2:3]
b <- c("1", "a", a )
length(b)
df <- data.frame(column_a = 1:8, column_b = b)
```

```
df[,1]
df$column_b
as.numeric(df$column_b)
plot(df)

df[1:3,]
df[,1]

as.factor(b)

d <- list(a, b, df)
d
names(d)
names(d) <- c("numeric_vector", "character_vector", "dataframe")
d
d[[1]]
d$numeric_vector

a == b
a != b

# 1. Set working directory ==================================================
setwd("C:/GIT/Digital-Soil-Mapping/")

# 2. Install and load packages ==============================================
# readxl, tidyverse, and data.table packages using the functions
install.packages("tidyverse")
install.packages("readxl")
install.packages("data.table")
library(tidyverse)
library(readxl)
library(data.table)

# 3. Import an spreadsheet ==================================================
## 3.1 Read the MS Excel file ---------------------------------------------
#Read the soil_data.xlsx file, spreadsheet 2, using read_excel
read_excel(path = "01-Data/soil_data.xlsx", sheet = 2)
```

```r
## 3.2 Read the csv file with the native function ------------------------------
# 01-Data/horizon.csv
read.csv("01-Data/soil_profile_data.csv")

## 3.3 Read the csv file with the tidyverse function ---------------------------
read_csv("01-Data/soil_profile_data.csv")

## 3.4 Read the csv file with the data.table function --------------------------
fread("01-Data/soil_profile_data.csv")

## 3.5 Assign the dataframe to an object called dat ----------------------------
dat <- read_csv("01-Data/soil_profile_data.csv")

# 4. Tidyverse functions ======================================================
## 4.1 Select pid, hip, top, bottom, ph_h2o, cec from dat ---------------------
dat_1 <- dat %>%
  select(id_prof, id_hor, top, bottom, ph_h2o, cec)

## 4.2 Filter: pick observations by their values -------------------------------
# filter observations with cec > 50 cmolc/100g

dat_2 <- dat_1 %>%
  filter(cec > 30)

dat_2
## 4.3 Mutate: create a new variable ------------------------------------------
# thickness = top - bottom

dat_3 <- dat_2 %>%
  mutate(thickness = bottom - top)

## 4.4 Group_by and summarise -------------------------------------------------
# group by variable pid
# summarise taking the mean of pH and cec

dat_4 <- dat_3 %>%
  group_by(id_prof) %>%
  summarise(mean_ph = mean(ph_h2o),
```

```
            mean_cec = mean(cec))

## 4.5 Reshape the table using pivot_longer ----------------------------------
# use dat_3
# put the names of the variables ph_h2o, cec and thickness in the column
# variable and keep the rest of the table. Save in dat_5

dat_5 <- dat_3 %>%
  pivot_longer(ph_h2o:thickness, names_to = "soil_property", values_to = "value")

## 4.6 Join the table sites.csv with dat_3 -----------------------------------
# Load soil_phys_data030.csv (in 01-Data folder)
# Join its columns with dat_3 keeping all the rows of dat_3
# save the result as dat_6

phys <- read_csv("01-Data/soil_phys_data030.csv")

phys <- phys %>% rename(id_prof = "ProfID")

dat_6 <- dat_3 %>%
  left_join(phys)
# or
dat_6 <- phys %>%
  right_join(dat_3)

# 5. Data visualization with ggplot2 ==========================================
## 5.1 1D plot: histograms ---------------------------------------------------
# histograms of cec and ph_h2o

ggplot(dat_3, aes(x=cec)) + geom_histogram()

## 5.2 2D plot: scatterplot --------------------------------------------------
# Scatterplot bottom vs. ph_h2o

ggplot(dat_3, aes(x = bottom, y = ph_h2o)) +
  geom_point()

# add a fitting line
```

```r
ggplot(dat_3, aes(x = bottom, y = ph_h2o)) +
  geom_point() +
  geom_smooth(method = "lm" )

## 5.3 3D plot: scatterplot ----------------------------------------------------
# Scatterplot bottom vs. ph_h2o, add clay as color and size inside the
# function aes()

ggplot(dat_3, aes(x = bottom, y = ph_h2o, color = cec, size = cec)) +
  geom_point()

# 6. Geospatial data with terra ================================================
## Load packages (install them if needed)
library(terra)
## 6.1 Load a raster and a vector layer ----------------------------------------
# Load 01-Data/covs/grass.tif using rast() function, then plot it
# Load 01-Data/soil map/SoilTypes.shp using vect() function and plot it
# explore the attributes of these layers

r <- rast("01-Data/Macedonia/grass.tif")
plot(r)

v <- vect("01-Data/Macedonia/SoilTypes.shp")
plot(v)

## 6.2 Load a raster and a vector layer ----------------------------------------
# Check the current CRS (EPSG) of the raster and the vector.
# Find a *projected* CRS in http://epsg.io for Macedonia and copy the number
# Check the Arguments of function project (?project) that need to be defined
# Save the new object as r_proj and v_proj
# plot both objects

r_proj <- project(x = r, y = "epsg:6204", method = "bilinear", res = 250)
plot(r_proj)
v_proj <- project(x = v, y = "epsg:6204")
plot(v_proj, add = TRUE)

## 6.3 Cropping and masking a raster -------------------------------------------
```

```r
# Compute the area of the polygons in v_proj (search for a function) and
# assign the values to a new column named area
# select the largest polygon using [], $, == and max() func. and save it as pol
# crop the raster with pol using the crop() function and save it as r_pol
# mask the raster r_pol with the polygon pol and save it with the same name
# plot each result

v_proj$area <- expanse(v_proj, unit = "ha")
pol <- v_proj[v_proj$area == max(v_proj$area)]
plot(pol)
r_pol <- crop(r_proj, pol)
plot(r_pol)
plot(pol, add = TRUE)
r_pol <- mask(r_pol, pol)
plot(r_pol)

## 6.4 Replace values in a raster by filtering their cells ---------------------
# Explore the following link to understand how terra manage cell values
# https://rspatial.org/terra/pkg/4-algebra.html
# Replace values lower than 5 in r+pol by 0

r_pol[r_pol$grass < 5] <- 0
plot(r_pol)

## 6.5 Rasterize a vector layer ------------------------------------------------
# Use rasterize() function to convert v_proj to raster
# Use r_proj as reference raster
# Use field Symbol to assign cell values, and plot the new map

v_class <- rasterize(x = v_proj, y = r_proj, field = "Symbol" )
plot(v_class)
v_class
activeCat(v_class) <- 1

## 6.6 Extracting raster values using points -----------------------------------
# Covert dat_6 to spatial points using vect() function (check help of vect())
# Note that the EPSG number is 6204
# Save the points as s
```

```r
# Plot s and r_proj together in the same map (Argument add=TRUE)
# Extract the values of the raster using extract() function (check the help)
# Remove the ID column of the extracted values
# merge the extracted data with s using cbind() function
# Convert s as a dataframe

s <- vect(dat_6, geom=c("x", "y"), crs = "epsg:6204")
plot(r_proj)
plot(s, add=TRUE)
x <- extract(r_proj,s, ID=FALSE)
s <- cbind(s,x)
d <- as.data.frame(s)
d
#GGally::ggscatmat(d)

## 6.7 Zonal statistics using polygons and rasters ----------------------------
# Use the extract() func. to estimate the mean value of r_proj at each polygon
# Use the fun= argument (check the help)
# Use the cbind() func. to merge v_proj and the extracted values
# convert v_proj to a dataframe
# Create a ggplot boxplot (geom_boxplot) with x=Symbol and y=grass

x <- extract(r_proj, v_proj, fun = mean, ID=FALSE)
v_proj <- cbind(v_proj, x)

d <- as_tibble(v_proj)

d %>%
  ggplot(aes(x =Symbol, y = grass, fill = Symbol)) +
  geom_boxplot() +
  ylab("Grass probability")

## 6.8 Bonus track: use zonal() function --------------------------------------

zonal(r_proj, v_class, na.rm=T)
```

Script 2: Data preparation

```
#
# Digital Soil Mapping
# Soil Profile Data
# Cleaning and Processing
#
# GSP-Secretariat
# Contact: Isabel.Luotto@fao.org
#          Marcos.Angelini@fao.org
#-------------------------------------------------------------------------------

#Empty environment and cache
rm(list = ls())
gc()

# Content of this script ======================================================
# The goal of this script is to organise the soil data for mapping, including:
#
# 0 - User-defined variables
# 1 - Set working directory and load necessary packages
# 2 - Import national data
# 3 - Select useful columns
# 4 - Quality check
# 5 - Estimate BD using pedotransfer function
# 6 - Harmonize soil layers
# 7 - Add chemical properties from additional dataset
# 8 - Plot and save results
#-------------------------------------------------------------------------------

# 0 - User-defined variables ==================================================
wd <- 'C:/Users/luottoi/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
#wd <- "C:/GIT/GSNmap-TM/Digital-Soil-Mapping"
#wd <- 'C:/Users/hp/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
#wd <- "/Users/luislado/Library/CloudStorage/Dropbox/GeoForsk/Clientes/FAO/GSNmap

# 1 - Set working directory and load necessary packages =======================
```

```r
setwd(wd) # change the path accordingly
#(setwd(dirname(rstudioapi::getActiveDocumentContext()$path)))

library(tidyverse) # for data management and reshaping
library(readxl) # for importing excel files
library(mapview) # for seeing the profiles in a map
library(sf) # to manage spatial data (shp vectors)
library(aqp) # for soil profile data
library(mpspline2) # for horizon harmonization
library(plotly) # interactive plots

# 2 - Import national data ========================================================
# Save your national soil dataset in the data folder /01-Data as a .csv file or
# as a .xlsx file

## 2.1 - for .xlsx files ------------------------------------------------------
# Import horizon data
# hor <- read_excel("01-Data/soil_data.xlsx", sheet = 2)
# # Import site-level data
# site <- read_excel("01-Data/soil_data.xlsx", sheet = 1)
# chem <- read_excel("01-Data/soil_data.xlsx", sheet = 2)
# phys <- read_excel("01-Data/soil_data.xlsx", sheet = 3)


## 2.2 - for .csv files -------------------------------------------------------
# Import horizon data
hor <- read_csv(file = "01-Data/soil_profile_data.csv",show_col_types = FALSE)
chem <- read_csv(file = "01-Data/soil_chem_data030.csv",show_col_types = FALSE)
phys <- read_csv(file = "01-Data/soil_phys_data030.csv",show_col_types = FALSE)


site <- select(hor, id_prof, x, y) %>% unique()
hor <- select(hor, id_prof, id_hor, top:cec)

# change names of key columns
names(site)
names(site)[1] <- "ProfID"
names(hor)
```

```r
names(hor)[1] <- "ProfID"
names(hor)[2] <- "HorID"
# scan the data
summary(site)
summary(hor)

# 3 - select useful columns ====================================================
## 3.1 - select AND RENAME columns ---------------------------------------------
hor <- select(hor, ProfID, HorID, top, bottom, ph=ph_h2o, k, soc, bd, cec)

# 4 - Quality check ============================================================

## 4.1 - Check locations -------------------------------------------------------
# Macedonian State Coordinate System EPSG:6204
# https://epsg.io/6204
site %>%
  st_as_sf(coords = c("x", "y"), crs = 4326) %>% # convert to spatial object
  mapview(zcol = "ProfID", cex = 3, lwd = 0.1) # visualise in an interactive map

site_sub <- site[site$ProfID=='2823',]

site_sub %>%
  st_as_sf(coords = c("x", "y"), crs = 4326) %>% # convert to spatial object
  mapview(zcol = "ProfID", cex = 3, lwd = 0.1) # visualise in an interactive map
# profile 2823 is wrongly located, so let's remove it
site <- filter(site, ProfID != 2823)


## 4.2 - Convert data into a Soil Profile Collection ---------------------------
aqp::depths(hor) <- ProfID ~ top + bottom
hor@site$ProfID <- as.numeric(hor@site$ProfID)
site(hor) <- left_join(site(hor), site)
profiles <- hor

profiles

## 4.3 - plot first 20 profiles using pH as color ------------------------------
plotSPC(x = profiles[1:20], name = "pH", color = "ph",
```

```
        name.style = "center-center")


## 4.4 - check data integrity ------------------------------------------------
# A valid profile is TRUE if all of the following criteria are false:
#    + depthLogic : boolean, errors related to depth logic
#    + sameDepth : boolean, errors related to same top/bottom depths
#    + missingDepth : boolean, NA in top / bottom depths
#    + overlapOrGap : boolean, gaps or overlap in adjacent horizons
aqp::checkHzDepthLogic(profiles)

# Identify non-valid profiles
dl <- checkHzDepthLogic(profiles)
dl[dl$depthLogic==T | dl$sameDepth==T | dl$missingDepth==T | dl$overlapOrGap==T,"

# visualize some of these profiles by the pid
subset(profiles, grepl(6566, ProfID, ignore.case = TRUE))
subset(profiles, grepl(7410 , ProfID, ignore.case = TRUE))
subset(profiles, grepl(8002, ProfID, ignore.case = TRUE))

## 4.5 - keep only valid profiles ------------------------------------------------
clean_prof <- HzDepthLogicSubset(profiles)
metadata(clean_prof)$removed.profiles
# write_rds(clean_prof, "01-Data/soilProfileCollection.rds")

## 4.6 convert soilProfileCollection to a table ---------------------------------
dat <- left_join(clean_prof@site, clean_prof@horizons)
dat <- select(dat, ProfID, HorID, x, y, top, bottom, ph:cec )


# 5 - Estimate BD using pedotransfer functions ================================

# create the function with all PTF
method_names <- c("Saini1996", "Drew1973", "Jeffrey1979", "Grigal1989",
                  "Adams1973", "Honeyset_Ratkowsky1989")

estimateBD <- function(SOC=NULL, method=NULL) {
  OM <- SOC * 1.724
```

```
  BD <- switch(method,
               "Saini1996" = 1.62 - 0.06 * OM,
               "Drew1973" = 1 / (0.6268 + 0.0361 * OM),
               "Jeffrey1979" = 1.482 - 0.6786 * (log(OM)),
               "Grigal1989" = 0.669 + 0.941 * exp(1)^(-0.06 * OM),
               "Adams1973" = 100 / (OM / 0.244 + (100 - OM) / 2.65),
               "Honeyset_Ratkowsky1989" = 1 / (0.564 + 0.0556 * OM),
               stop("Invalid method specified.")
  )
  return(BD)
}

# 5.1 - Select a pedotransfer function ----------------------------------------


# Create a test dataset with BD and SOC data
BD_test <- data.frame(SOC = dat$soc, BD_observed = dat$bd)

# Remove missing values
BD_test <- BD_test[complete.cases(BD_test),]
BD_test <- na.omit(BD_test)


# 5.2 - Estimate BLD for a subset using the pedotransfer functions ------------
for (i in method_names) {
  BD_test[[i]] <- estimateBD(BD_test$SOC, method = i)
}

# Print the resulting data frame
BD_test


## 5.3 Compare results ---------------------------------------------------------
# Observed values:
summary(BD_test)

# Compare data distributions for observed and predicted BLD
plot.bd <- BD_test %>%
```

```
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed", "Saini1996", "Drew1973", "Jeffrey1979",
                        "Grigal1989", "Adams1973", "Honeyset_Ratkowsky1989"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density()

plot.bd

# Dymanic plot with plotly
ggplotly(plot.bd)

ggplotly(plot.bd) %>%
  layout(hovermode = "x")

# Plot the Selected function again
BD_test %>%
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed", "Honeyset_Ratkowsky1989"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density() + xlim(c(0,2.5))

# Same dynamic plot
ggplotly(BD_test %>%
           select(-SOC) %>%
           pivot_longer(cols = c("BD_observed", "Honeyset_Ratkowsky1989"),
                        names_to = "Method", values_to = "BD") %>%
           ggplot(aes(x = BD, color = Method)) +
           geom_density() + xlim(c(0,2.5))) %>%
  layout(hovermode = "x")

## 5.4 Estimate BD for the missing horizons ----------------------------------
dat$bd[is.na(dat$bd)] <-
  estimateBD(dat[is.na(dat$bd),]$soc, method="Honeyset_Ratkowsky1989")

# Explore the results
summary(dat$bd)
```

```
g <- BD_test %>%
  select(-SOC) %>%
  pivot_longer(cols = c("BD_observed"),
               names_to = "Method", values_to = "BD") %>%
  ggplot(aes(x = BD, color = Method)) +
  geom_density() +
  xlim(c(0,2.5))
g + geom_density(data = dat, aes(x=bd, color = "Predicted +\n observed"))


## 5.5 - Explore outliers ----------------------------------------------------
# Outliers should be carefully explored and compared with literature values.
# Only if it is clear that outliers represent impossible or highly unlikely
# values, they should be removed as errors.
#
# Carbon content higher than 15% is only typical for organic soil (histosols)
# We will remove all atypically high SOC as outliers
summary(dat$soc)
na.omit(dat$ProfID[dat$soc > 10])
dat$ProfID[dat$soc > 10][!is.na(dat$ProfID[dat$soc > 10])]

dat <- dat[dat$ProfID != 6915,]
dat <- dat[dat$ProfID != 7726,]

dat<- dat[!(dat$ProfID %in% dat$ProfID[dat$soc > 10][!is.na(dat$ProfID[dat$soc >

# Explore bulk density data, identify outliers
# remove layers with Bulk Density < 1 g/cm^3
low_bd_profiles <- na.omit(dat$ProfID[dat$bd<1])
dat <- dat[!(dat$ProfID %in% low_bd_profiles),]

# Explore data, identify outliers
x <- pivot_longer(dat, cols = ph:cec, values_to = "value",
                  names_to = "soil_property")
x <- na.omit(x)
ggplot(x, aes(x = soil_property, y = value, fill = soil_property)) +
  geom_boxplot() +
  facet_wrap(~soil_property, scales = "free")
```

```r
# 6 - Harmonize soil layers ============================================
source("03-Scripts/.spline_functions.R")
## 6.1 - Set target soil properties and depths ---------------------------
names(dat)
dat <- select(dat, ProfID, HorID, x, y, top, bottom, ph, k, soc, bd, cec)

target <- c("ph", "k", "soc",  "bd", "cec")
depths <- c(0,30)

## 6.2 - Create standard layers ------------------------------------------
splines <- apply_mpspline_all(df = dat, properties = target, depth_range = depths
summary(splines)

# merge splines with x and y
d <- unique(select(dat, ProfID, x, y))
d <- left_join(d, splines)



# 7 - Harmonize units ==================================================
#Harmonize units if different from target units
# Mandatory Soil Propertes and corresponing units:
# Total N - ppm
# Available P - ppm
# Available K - ppm
# Cation exchange capacity cmolc/kg
# pH
# SOC - %
# Bulk density g/cm3
# Soil fractions (clay, silt and sand) -

# Units soil profile data (dataframe d)
#
head(d) # pH; K cmolc/kg; SOC %; BD g/cm3; CEC  cmolc/kg

# K => convert cmolc/kg to ppm (K *10 * 39.096)
d$k_0_30 <- d$k_0_30 * 10 * 39.096
```

```r
head(chem)# P ppm; N %; K ppm
# N => convert % to ppm (N * 10000)
chem$tn <-chem$tn*10000

head(phys)# clay, sand, silt g/kg
# convert g/kg to % (/10)
phys$clay_0_30 <-phys$clay_0_30/10
phys$sand_0_30  <-phys$sand_0_30 /10
phys$silt_0_30 <-phys$silt_0_30/10

# Correct negative clay content
phys$clay_0_30[phys$clay_0_30<0] <- 0


# Add chemical and physical properties from additional datasets ================


# Rename columns to match the main data set
names(d)
names(chem)[1] <- 'ProfID'
names(chem)[4] <- 'p_0_30'
names(chem)[5] <- 'k_0_30'
names(chem)[6] <- 'n_0_30'


#The chem dataframe comes from and independent dataset we need to create new uniq
#Create unique ProfID
chem$ProfID <- seq(max(d$ProfID)+1,max(d$ProfID)+1+nrow(chem)-1)

# Add the new data as new rows using dplyr we can add empty rows
# automatically for the not measured properties in the chem dataset
d <- bind_rows(d, chem)

#The phys dataframe with the texture instead shares the same ProfIDs (we can dire
d <- left_join(d, phys, by=c('ProfID', 'x', 'y'))

# 8 - Plot  and save results ===================================================
names(d)
```

```
x <- pivot_longer(d, cols = ph_0_30:silt_0_30, values_to = "value",
                  names_sep = "_",
                  names_to = c("soil_property", "top", "bottom"))
x <- mutate(x, depth = paste(top, "-" , bottom))
#x <- na.omit(x)
ggplot(x, aes(x = depth, y = value, fill = soil_property)) +
  geom_boxplot() +
  facet_wrap(~soil_property, scales = "free")


ggplotly(ggplot(x, aes(x = depth, y = value, fill = soil_property)) +
           geom_boxplot() +
           facet_wrap(~soil_property, scales = "free"))

# save the data
write_csv(d, "02-Outputs/harmonized_soil_data.csv")
```

Scripts 3: Download environmental covariates

```
var assets = ["projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio1",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio12",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio13",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio14",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio16",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio17",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio5",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/bio6",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/ngd10",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_max",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_mean",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_min",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/pet_penman_range",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_max",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_mean",
"projects/digital-soil-mapping-gsp-fao/assets/CHELSA/sfcWind_range",
```

```
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_030405_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_030405_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_060708_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_060708_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_091011_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_091011_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_120102_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/fpar_120102_500m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_030405_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_030405_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_060708_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_060708_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_091011_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_091011_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_120102_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/lstd_120102_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_030405_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_030405_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_060708_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_060708_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_091011_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_091011_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_120102_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndlst_120102_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_030405_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_030405_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_060708_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_060708_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_091011_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_091011_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_120102_250m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/ndvi_120102_250m_sd",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/snow_cover",
"projects/digital-soil-mapping-gsp-fao/assets/MODIS/swir_060708_500m_mean",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/crops",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/flooded_vegetation",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/grass",
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/shrub_and_scrub",
```

```
"projects/digital-soil-mapping-gsp-fao/assets/LANDCOVER/trees",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_curvature_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_downslopecurvature_
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_dvm2_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_dvm_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_elevation_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_mrn_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_neg_openness_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_pos_openness_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_slope_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_tpi_250m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_twi_500m",
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_upslopecurvature_25
"projects/digital-soil-mapping-gsp-fao/assets/OPENLANDMAP/dtm_vbf_250m"];

// Load borders

/// Using LSIB 2017 (replace the countries that you want to download)
var country_list = ['Italy'];
var aoi = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
  .filter(ee.Filter.inList('country_na', country_list));
var region = aoi.geometry();

/// Using a shapefile
/// 1. Upload the borders of your countries as an asset
/// 2. Replace 'your_shapefile' with the path to your shapefile
// var shapefile = ee.FeatureCollection('users/your_username/your_shapefile');
// var region = shapefile.geometry();


// Load assets as ImageCollection
var assetsCollection = ee.ImageCollection(assets);

// Clip each image in the collection to the region of interest
var clippedCollection = assetsCollection.map(function(img){
  return img.clip(region).toFloat();
});
```

```
// Function to replace masked values with zeroes for fpar bands
function replaceMaskedFpar(img) {
  var allBands = img.bandNames();
  var fparBands = allBands.filter(ee.Filter.stringStartsWith('item', 'fpar'));
  var nonFparBands = allBands.removeAll(fparBands);

  var fparImg = img.select(fparBands).unmask(0);
  var nonFparImg = img.select(nonFparBands);

  // If there are no fpar bands, return the original image
  var result = ee.Algorithms.If(fparBands.length().eq(0),
                                img,
                                nonFparImg.addBands(fparImg));

  return ee.Image(result);
}

// Clip each image in the collection to the region of interest and replace masked
var clippedCollection = assetsCollection.map(function(img){
  var clippedImg = img.clip(region).toFloat();
  return replaceMaskedFpar(clippedImg);
});

// Stack the layers and maintain the layer names in the final file
var stacked = clippedCollection.toBands();

// Get the list of asset names
var assetNames = ee.List(assets).map(function(asset) {
  return ee.String(asset).split('/').get(-1);
});

// Rename the bands with asset names
var renamed = stacked.rename(assetNames);
print(renamed, 'Covariates to be exported')

// Visualize the result
// Set a visualization parameter (you can adjust the colors as desired)
var visParams = {
```

```
  bands: 'bio1',
  min: 0,
  max: 1,
  palette: ['blue', 'green', 'yellow', 'red']
};

// Add the layer to the map
Map.centerObject(renamed, 6)
Map.addLayer(renamed, visParams, 'Covariates');
// Export the stacked image to Google Drive
Export.image.toDrive({
  image: renamed,
  description: 'covariates',
  folder: 'GEE',
  scale: 250,
  maxPixels: 1e13,
  region: region
});

/* Create mask for croplands ----------------------------*/

// Load the Copernicus Global Land Service image collection
var imageCollection = ee.Image("COPERNICUS/Landcover/100m/Proba-V-C3/Global/2019"
  .select("discrete_classification")
  .clip(region)

var crs = 'EPSG:4326'; // WGS84
var res = 250; // Resolution in decimal degrees

// Default resampling is nearest neighbor
var image1 = imageCollection.resample()
  .reproject({
    crs: crs, // Add your desired CRS here
    scale: res // Add your desired scale here
  });

// Reclassify the land cover classes
var inList = [0, 20, 30, 40, 50, 60, 70, 80, 90, 100, 111, 112, 113, 114, 115, 11
```

```
              121, 122, 123, 124, 125, 126, 200];
var outList = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

var FAO_lu = image1.remap(inList, outList)
  .toDouble()
  .clip(region);

// print(FAO_lu)

// Convert 0 to NA
var mask = FAO_lu.neq(0);
print(mask)
FAO_lu = FAO_lu.updateMask(mask);

print(FAO_lu, "Mask")

var visParams = {
  bands: 'remapped',
  min: 0,
  max: 1,
  palette: ['green', 'yellow']
};

// Add the layer to the map
Map.addLayer(FAO_lu,visParams ,'Mask');

// Export the land cover image as a raster to Google Drive
Export.image.toDrive({
  image: FAO_lu,
  folder: 'GEE',
  description: 'mask',
  scale: res, // Add your desired scale here
  region: region,
  crs: crs, // Add your desired CRS here
  maxPixels: 1e13 // Add a maximum number of pixels for export if needed
});
```

```
# Create terrain derivatives from MERIT-DEM and
# harmonise their format with the rest of the covariates
# Marcos Angelini, Luis Rodríguez Lado & Isabel Luotto
# Global Soil Partnership - FAO
rm(list = ls())
setwd("C:/GIT/GSNmap-TM/Digital-Soil-Mapping/")

# Load libraries
  library(terra)
  library(raster)

# import high resolution raster
dem <- rast("01-Data/create_covariates/MERIT_DEM.tif")

# Plot DEM
plot(dem)

# Obtain DEM derived maps
derived_vars <- terrain(dem,
                        c("slope", "roughness", "aspect", "flowdir",
                          "aspect","TPI", "TRI", "TRIriley", "TRIrmsd"),
                        unit = "degrees")

plot(derived_vars, nc=3, nr=3)

# load rster of reference
ref <- rast("01-Data/covs/Covariates.tif")[[1]]

derived_vars
ref
# transform the variable's coordinates, resolution and extent
resampled_vars <- project(derived_vars, ref)
# mask variables
resampled_vars <- mask(resampled_vars, ref)
plot(resampled_vars)

resampled_vars
ref
```

```
# save the rasters
writeRaster(resampled_vars, "01-Data/covs/dem_derivatives.tif")

# # Load a vector layer (e.g.a soil map)
# v <- vect("01-Data/AOI.shp")
# plot(v, "shape_Area")
# # Vector rasterization with continuous data
# x <- rasterize(x = v, y = ref, field = "shape_Area", fun = "max")
# plot(x)
# y <- rasterize(x = v, y = ref, field = "CABECERA")
#
# writeRaster(y, "01-Data/create_covariates/polygon_map.tif")
```

Script 4: Modelling, validation and prediction using soil data with coordinates

```
#_______________________________________________________________________________
#
# Quantile Regression Forest
# Soil Property Mapping
#
# GSP-Secretariat
# Contact: Isabel.Luotto@fao.org
#          Marcos.Angelini@fao.org
#_______________________________________________________________________________

#Empty environment and cache
rm(list = ls())
gc()

# Content of this script ======================================================
# 0 - Set working directory, soil attribute, and packages
# 1 - Merge soil data with environmental covariates
# 2 - Covariate selection
# 3 - Model calibration
```

```r
# 4 - Uncertainty assessment
# 5 - Prediction
# 6 - Export final maps
#-------------------------------------------------------------------------------


# 0 - Set working directory, soil attribute, and packages ======================

# Working directory
wd <- 'C:/Users/luottoi/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
setwd(wd)

# Define country of interes throuhg 3-digit ISO code
ISO ='ISO'

# Load Area of interest (shp)
AOI <- '01-Data/AOI.shp'

# Terget soil attribute (Mandatory 10)
target_properties<- c("ph_0_30", "k_0_30" , "soc_0_30" ,"bd_0_30", "cec_0_30","p_
                      "n_0_30","clay_0_30", "sand_0_30" ,"silt_0_30")

# Function for Uncertainty Assessment
load(file = "03-Scripts/eval.RData")

#load packages
library(tidyverse)
library(data.table)
library(caret)
library(quantregForest)
library(terra)
library(sf)
library(doParallel)


# 1 - Merge soil data with environmental covariates ============================

## 1.1 - Load covariates -------------------------------------------------------
```

```r
files <- list.files(path= '01-Data/covs/', pattern = '.tif$', full.names = T)
ncovs <- list.files(path= '01-Data/covs/', pattern = '.tif$', full.names = F)
#In case of extent error, or if covariates other than the default ones are added
# ref <- rast(files[1])
# covs <- list()
# for (i in seq_along(files)) {
#   r <- rast(files[i])
#   r <- project(r, ref)
#   covs[[i]] <- r
# }
# covs <- rast(covs)

covs<- rast(files)
ncovs <-  filename <- sub('.tif', '', ncovs)

ncovs[ncovs=="dtm_neg-openness_250m"] = 'dtm_neg'
ncovs[ncovs=="dtm_pos-openness_250m"] = 'dtm_pos'
names(covs) <- ncovs


## 1.2 - Load the soil data (Script 2) ----------------------------------------
dat <- read_csv("02-Outputs/harmonized_soil_data.csv")

# Convert soil data into a spatial object (check https://epsg.io/6204)
dat <- vect(dat, geom=c("x", "y"), crs = crs(covs))

# Reproject point coordinates to match coordinate system of covariates
dat <- terra::project(dat, covs)
names(dat)

## 1.3 - Extract values from covariates to the soil points --------------------
pv <- terra::extract(x = covs, y = dat, xy=F)
dat <- cbind(dat,pv)
dat <- as.data.frame(dat)

summary(dat)
```

```r
for(soilatt in unique(target_properties)){


## 1.4 - Target soil attribute + covariates ---------------------------------
d <- dplyr::select(dat, soilatt, names(covs))
d <- na.omit(d)

# 2 - Covariate selection with RFE ============================================
## 2.1 - Setting parameters ------------------------------------------------
# Repeatedcv = 3-times repeated 10-fold cross-validation
fitControl <- rfeControl(functions = rfFuncs,
                         method = "repeatedcv",
                         number = 10,          ## 10 -fold CV
                         repeats = 3,         ## repeated 3 times
                         verbose = TRUE,
                         saveDetails = TRUE,
                         returnResamp = "all")

# Set the regression function
fm = as.formula(paste(soilatt," ~", paste0(ncovs,
                                           collapse = "+")))

# Calibrate the model using multiple cores
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)


## 2.2 - Calibrate a RFE model to select covariates ----------------------------
covsel <- rfe(fm,
              data = d,
              sizes = seq(from=10, to=length(ncovs)-1, by = 5),
              rfeControl = fitControl,
              verbose = TRUE,
              keep.inbag = T)
stopCluster(cl)
saveRDS(covsel, "02-Outputs/models/covsel.rda")

## 2.3 - Plot selection of covariates ----------------------------------------
```

```
trellis.par.set(caretTheme())
plot(covsel, type = c("g", "o"))

# Extract selection of covariates and subset covs
opt_covs <- predictors(covsel)

# 3 - QRF Model calibration ===================================================
## 3.1 - Update formula with the selected covariates ---------------------------
fm <- as.formula(paste(soilatt," ~", paste0(opt_covs, collapse = "+")))

# parallel processing
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)

## 3.2 - Set training parameters -----------------------------------------------
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,          ## 10 -fold CV
                           repeats = 3,         ## repeated 3 times
                           savePredictions = TRUE)

# Tune mtry hyperparameters
mtry <- round(length(opt_covs)/3)
tuneGrid <-  expand.grid(mtry = c(mtry-5, mtry, mtry+5))

## 3.3 - Calibrate the QRF model -----------------------------------------------
model <- caret::train(fm,
                      data = d,
                      method = "qrf",
                      trControl = fitControl,
                      verbose = TRUE,
                      tuneGrid = tuneGrid,
                      keep.inbag = T,
                      importance = TRUE)
stopCluster(cl)
gc()


## 3.4 - Extract predictor importance as relative values (%)
```

```
x <- randomForest::importance(model$finalModel)
model$importance <- x
## 3.5 - Print and save model ----------------------------------------------------
print(model)
saveRDS(model, file = paste0("02-Outputs/models/model_",soilatt,".rds"))

# 4 - Uncertainty assessment ==================================================
# extract observed and predicted values
o <- model$pred$obs
p <- model$pred$pred
df <- data.frame(o,p)

## 4.1 - Plot and save scatterplot -----------------------------------------------
(g1 <- ggplot(df, aes(x = o, y = p)) +
  geom_point(alpha = 0.1) +
   geom_abline(slope = 1, intercept = 0, color = "red")+
  ylim(c(min(o), max(o))) + theme(aspect.ratio=1)+
  labs(title = soilatt) +
  xlab("Observed") + ylab("Predicted"))
# ggsave(g1, filename = paste0("02-Outputs/residuals_",soilatt,".png"), scale = 1
#        units = "cm", width = 12, height = 12)

## 4.2 - Print accuracy coeficients ----------------------------------------------
# https://github.com/AlexandreWadoux/MapQualityEvaluation
eval(p,o)

## 4.3 - Plot Covariate importance -----------------------------------------------
(g2 <- varImpPlot(model$finalModel, main = soilatt, type = 1))

# png(filename = paste0("02-Outputs/importance_",soilatt,".png"),
#     width = 15, height = 15, units = "cm", res = 600)
# g2
# dev.off()

# 5 - Prediction ==============================================================
# Generation of maps (prediction of soil attributes)
## 5.1 - Produce tiles -----------------------------------------------------------
r <-covs[[1]]
```

```
t <- rast(nrows = 5, ncols = 5, extent = ext(r), crs = crs(r))
tile <- makeTiles(r, t,overwrite=TRUE,filename="02-Outputs/tiles/tiles.tif")

## 5.2 - Predict soil attributes per tiles ------------------------------------
# loop to predict on each tile

for (j in seq_along(tile)) {
  gc()
  t <- rast(tile[j])
  covst <- crop(covs, t)


  # plot(r)#
  pred_mean <- terra::predict(covst, model = model$finalModel, na.rm=TRUE,
                              cpkgs="quantregForest", what=mean)
  pred_sd <- terra::predict(covst, model = model$finalModel, na.rm=TRUE,
                            cpkgs="quantregForest", what=sd)



  # ###### Raster package solution (in case terra results in many NA pixels)
  # library(raster)
  # covst <- stack(covst)
  # class(final_mod$finalModel) <-"quantregForest"
  # # Estimate model uncertainty
  # pred_sd <- predict(covst,model=final_mod$finalModel,type=sd)
  # # OCSKGMlog prediction based in all available data
  # pred_mean <- predict(covst,model=final_mod)
  #
  #
  # ###################################

  writeRaster(pred_mean,
              filename = paste0("02-Outputs/tiles/soilatt_tiles/",
                                soilatt,"_tile_", j, ".tif"),
              overwrite = TRUE)
  writeRaster(pred_sd,
              filename = paste0("02-Outputs/tiles/soilatt_tiles/",
```

```
                                soilatt,"_tileSD_", j, ".tif"),
              overwrite = TRUE)

  rm(pred_mean)
  rm(pred_sd)


  print(paste("tile",tile[j]))
}

## 5.3 - Merge tiles both prediction and st.Dev --------------------------------
f_mean <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                     pattern = paste0(soilatt,"_tile_"), full.names = TRUE)
f_sd <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                   pattern =  paste0(soilatt,"_tileSD_"), full.names = TRUE)
r_mean_l <- list()
r_sd_l <- list()

for (g in 1:length(f_mean)){
  r <- rast(f_mean[g])
  r_mean_l[g] <-r
  rm(r)
}

for (g in 1:length(f_sd)){

  r <- rast(f_sd[g])
  r_sd_l[g] <-r
  rm(r)
}
r_mean <-sprc(r_mean_l)
r_sd <-sprc(r_sd_l)
pred_mean <- mosaic(r_mean)
pred_sd <- mosaic(r_sd)

aoi <- vect(AOI)
pred_mean <- mask(pred_mean,aoi)
pred_sd <- mask(pred_sd,aoi)
```

```
plot(pred_mean)
plot(pred_sd)


# 6 - Export final maps ========================================================
## 6.1 - Mask croplands ------------------------------------------------------
msk <- rast("01-Data/mask.tif")
plot(msk)
pred_mean <- mask(pred_mean, msk)
plot(pred_mean)
pred_sd <- mask(pred_sd, msk)
plot(pred_sd)
plot(pred_sd/pred_mean*100, main = paste("CV",soilatt))

## 6.2 - Save results --------------------------------------------------------

# Harmonized naming
if (soilatt == 'ph_0_30'){
  name <-'_GSNmap_pH_Map030.tiff'
}else if (soilatt == 'k_0_30'){
  name <-'_GSNmap_Ktot_Map030.tiff'
}else if (soilatt == 'soc_0_30'){
  name <-'_GSNmap_SOC_Map030.tiff'
}else if (soilatt == 'clay_0_30'){
  name <-'_GSNmap_Clay_Map030.tiff'
}else if (soilatt == 'bd_0_30'){
  name <-'_GSNmap_BD_Map030.tiff'
}else if (soilatt == 'cec_0_30'){
  name <-'_GSNmap_CEC_Map030.tiff'
}else if (soilatt == 'p_0_30'){
  name <-'_GSNmap_Pav_Map030.tiff'
}else if (soilatt == 'n_0_30'){
  name <-'_GSNmap_Ntot_Map030.tiff'
}else if (soilatt == 'sand_0_30'){
  name <-'_GSNmap_Sand_Map030.tiff'
}else if (soilatt == 'silt_0_30'){
  name <-'_GSNmap_Silt_Map030.tiff'
```

```
}

writeRaster(pred_mean,
            paste0("02-Outputs/maps/",ISO,name),
            overwrite=TRUE)
writeRaster(pred_sd,
            paste0("02-Outputs/maps/",ISO, '_SD',name),
            overwrite=TRUE)



}
```

Annex II: R scripts for extra functions

Script to estimate Organic Carbon Stock

```
# Estimate Organic Carbon Stock (ocs) ====================================
# SOC must be in g/kg (% * 10)
# BLD in kg/m3
# CRF in percentage
d <- read_csv("02-Outputs/spline_soil_profile.csv")
# 0 - 30 cm
ORCDRC <- d$soc_0_30*10
HSIZE <- 30
BLD <- d$bd_0_30*1000
CRFVOL <- d$crf_0_30

OCSKG_0_30 <- ORCDRC/1000 * HSIZE/100 * BLD * (100 - CRFVOL)/100

# Convert Organic Carbon Stock from kg/m3 to t/ha
d$ocs_0_30 <- OCSKG_0_30*10
```

Annex III: Mapping without point coordinates

This chapter provides step-by-step instructions in case the soil measurements do not have point coordinates but come with information on the sampling region, i.e. location within an administrative unit. In this approach, soil samples are randomly distributed across the respective area (see Figure 10.1).

The steps are similar to the ones specified in Step 3. After merging the soil and environmental covariate data, the ones with most explanatory power are selected for model calibration. The uncertainty model is then assessed and finally the target soil property predicted. This is followed by the exporting of the final map products.

First, the working directory is set, the file path to the shapefile that contains the area of interest as well as the name of the target soil property are assigned to two object. Next, the number of simulations in which the samples are repeatedly distributed by random over the respective area is specified (see Figure 10.1). Next, the saved function for uncertainty assessment is loaded as well as all packages that are needed to run the script.

```
# 0 - Set working directory, soil attribute, and packages ======================

# Working directory
setwd('C:/GIT/GSNmap-TM/Digital-Soil-Mapping')

# Load Area of interest (shp)
AOI <- '01-Data/AOI_Arg.shp'
```

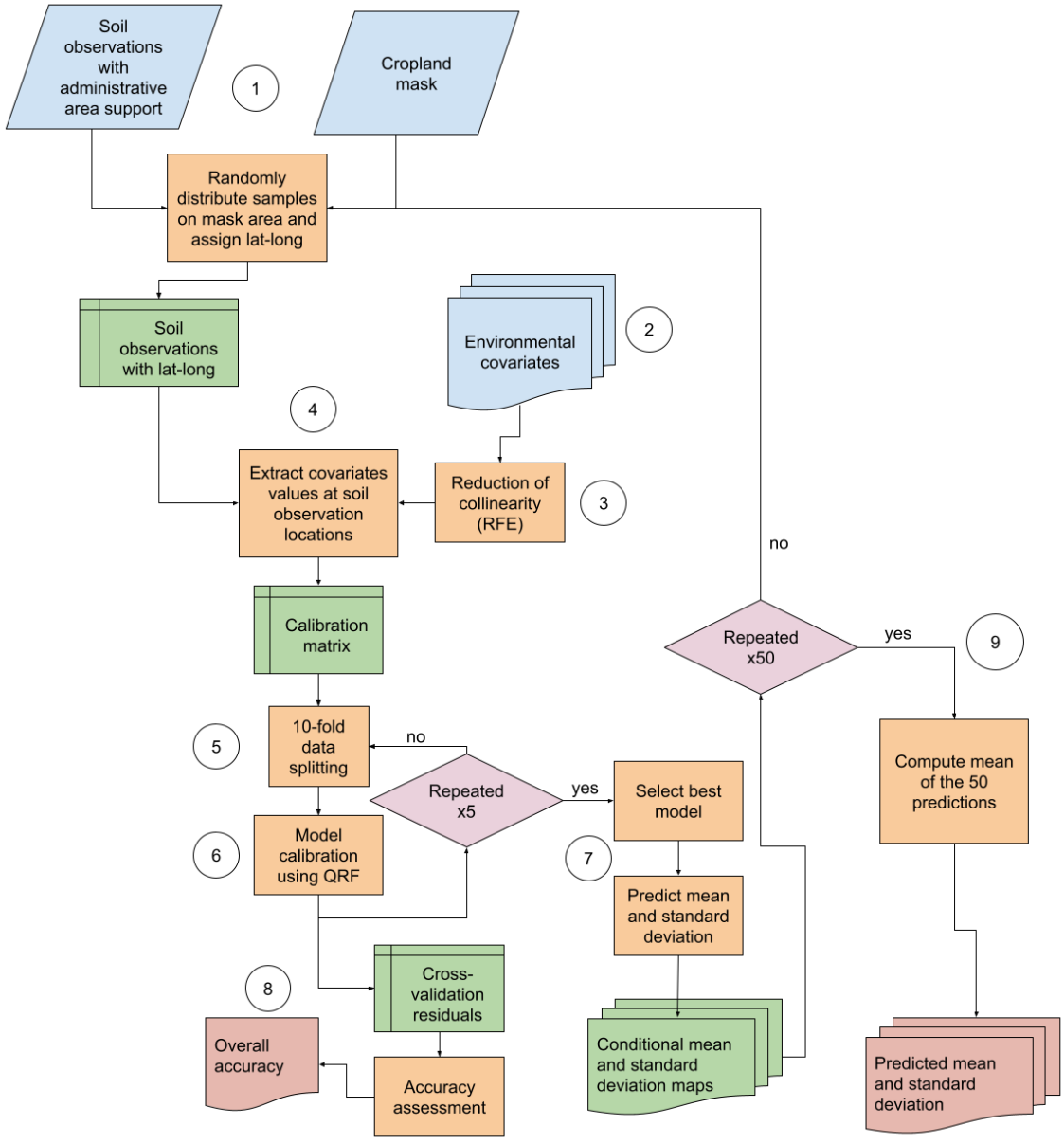


Figure 10.1: Digital soil mapping approach for area-support data. Circles are the steps.

```
# Terget soil attribute
soilatt <- 'p_bray'

# Repetitions (should be equal or greater than 50)
n_sim = 5

# Function for Uncertainty Assessment
load(file = "03-Scripts/eval.RData")

#load packages
library(tidyverse)
library(data.table)
library(caret)
library(quantregForest)
library(terra)
library(sf)
library(doParallel)
library(mapview)
# install.packages("terra", repos = "https://rspatial.r-universe.dev")
```

In the following lines of code, the data for validation is loaded. It consists of soil data that has point coordinates. However, if no data with covariates is available, this step can be skipped. Also, environmental covariate data are loaded and stacked together into one R object. Of this R object "covs", values are extracted for the soil profile locations of the validation data. Finally, the data without coordinates are loaded.

```
# 1 - Load validation data (with coordinates) if available ======================

val_data <- read_csv("01-Data/data_with_coord.csv") %>%
  vect(geom=c("x", "y"), crs = "epsg:4326")

## 1.1 - Load covariates ------------------------------------------------------
# Single-band files
f <- list.files("01-Data/covs/", ".tif$", full.names = TRUE)
f <- f[f !="01-Data/covs/covs.tif"]
covs <- rast(f)
```

```
# Multi-band file
# covs <- rast("01-Data/covs/covs.tif")

# extract names of covariates
ncovs <- names(covs)
ncovs <- str_replace(ncovs, "-", "_")
names(covs) <- ncovs

## 1.2 - extract values from covariates for validation data ---------------------
ex <- terra::extract(x = covs, y = val_data, xy=F)
val_data <- as.data.frame(val_data)
val_data <- cbind(val_data, ex)
val_data <- na.omit(val_data)

## 1.3 - Load the soil data without coordinates ---------------------------------
dat <- read_csv("01-Data/Buenos_Aires_sur_P_Bray.csv")
```

The next step groups the sample by "*stratum*", i.e. district or other administrative unit level. The number of samples per stratum are counted and only strata with more than 5 samples retained.

```
# Compute the number of samples per stratum
N <- dat %>% group_by(stratum) %>%
  summarise(N=n()) %>%
  na.omit()
# remove any stratum with few samples
N <- N[N$N > 5,]

# null dataframe to store model residuals
df <- NULL
```

In the following lines of code the modelling of the target soil attribute is modelled for each stratum. First, each stratum is masked with a cropland layer and then the number of pixels is counted. Next, a number of pixels is extracted from the stratum that corresponds to the total number of samples for the area. Within another for loop, the number of sites per stratum are selected and the user can check the exact location. Then, the selected coordinates are combined

with the soil data without coordinates and the values of the environmental covariates for these points are extracted with the extract() function of the terra package. After formulating the model formula, the model is trained and then calibrated using parallel computing. Note that cross-validation is not applied. Variable importance of the covariates is assessed and the model is saved. The uncertainty assessment uses the validation data (if available) as observed values and the predicted values to calculated the error indices and model fit parameters explained in the Section on Uncertainty assessment. Finally, the point values are plotted as scatterplots and residuals are saved as a .csv file.

```
# 2 - Sequentially model the soil attribute ====================================
for (i in 1:n_sim) {
  # load tif file of strata (masked by croplands)
  stratum <- rast("01-Data/land cover/SE_districts_croplands.tif")
  # Compute the number of pixels within each stratum
  y <- freq(stratum)
  # randomly select max(N) pixels within each stratum
  x <- spatSample(x = stratum, max(N$N), method = "stratified",
                  as.df=T, xy=T, replace=F)
  table(x$stratum)
  # remove any stratum that is not present in the data (dat)
  x <- x[x$stratum %in% N$stratum,]
  # load data without coordinates again
  dat <- read_csv("01-Data/Buenos_Aires_sur_P_Bray.csv") %>%
    na.omit()
  # create a vector with strata
  strata <- unique(x$stratum)
  # select the corresponding number of sites per stratum
  d <- NULL
  for (j in seq_along(strata)) {
    z <- x[x$stratum==strata[j],]
    n <- N[N$stratum==strata[j],"N"][[1]]
    z <- z[sample(1:max(N$N), size = n),]
    if(strata[j] %in% dat$stratum){
      z <- cbind(z,dat[dat$stratum==strata[j],soilatt])
      d <- rbind(d,z)}
  }
  # check the distribution of points
  d %>%
```

```
  st_as_sf(coords = c("x", "y"), crs = 4326) %>% # convert to spatial object
  mapview(zcol = soilatt, cex = 2, lwd = 0.1) #+ mapview(raster(stratum))
dat <- vect(d, geom=c("x", "y"), crs = "epsg:4326")

# Extract values from covariates
pv <- terra::extract(x = covs, y = dat, xy=F)
dat <- cbind(dat,pv)
dat <- as.data.frame(dat)

# Select from dat the soil attribute and the covariates and remove NAs
d <- select(dat, soilatt, ncovs)
d <- na.omit(d)

# formula
fm = as.formula(paste(soilatt," ~", paste0(ncovs,
                                           collapse = "+")))
# parallel processing
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)

# Set training parameters (CV is not applied)
fitControl <- trainControl(method = "none",
                           # number = 10,          ## 10 -fold CV
                           # repeats = 3,         ## repeated 3 times
                           savePredictions = TRUE)

# Tune mtry hyperparameter
mtry <- round(length(ncovs)/3)
tuneGrid <-  expand.grid(mtry = c(mtry))

# Calibrate the QRF model
print(paste("start fitting model", i))
model <- caret::train(fm,
                      data = d,
                      method = "qrf",
                      trControl = fitControl,
                      verbose = TRUE,
                      tuneGrid = tuneGrid,
```

```
                        keep.inbag = T,
                        importance = TRUE, )
stopCluster(cl)
gc()
print(paste("end fitting", i))
# Extract predictor importance
x <- randomForest::importance(model$finalModel)
model$importance <- x
# Plot Covariate importance
(g2 <- varImpPlot(model$finalModel, main = soilatt, type = 1))
# Save the plot if you like
# png(filename = paste0("02-Outputs/importance_",soilatt,"_",i,".png"),
#     width = 15, height = 15, units = "cm", res = 600)
# g2
# dev.off()
# Print and save model
print(model)
saveRDS(model, file = paste0("02-Outputs/models/model_",soilatt,"_",i,".rds"))

# Uncertainty assessment
# extract observed and predicted values
o <- val_data[,soilatt]
p <- predict(model$finalModel, val_data, what = mean)
df <- rbind(df, data.frame(o,p, model=i))
# Print accuracy coeficients
# https://github.com/AlexandreWadoux/MapQualityEvaluation
print(paste("model", i))
print(eval(p,o))

# Plot and save scatterplot
(g1 <- ggplot(df, aes(x = o, y = p)) +
   geom_point(alpha = 0.5) +
   geom_abline(slope = 1, intercept = 0, color = "red")+
   ylim(c(min(o), max(o))) + theme(aspect.ratio=1)+
   labs(title = soilatt) +
   xlab("Observed") + ylab("Predicted"))

# save the plots if you like
```

```
  # ggsave(g1, filename = paste0("02-Outputs/residuals_",soilatt,"_",i,".png"),
  #        scale = 1,
  #        units = "cm", width = 12, height = 12)

}
write_csv(df, paste("residuals_",soilatt,".csv"))
```

The previous model fitting and calibration is simulated 5 times, as specified at the beginning of the script. In the following part of the script, the conditional mean and standard deviation are predicted for each simulation. For that, the study area is divided into tiles. For each tile, the respective covariates and models (saved in the previous step) are used to predict mean and standard deviation which are then stored in two separate raster files.

```
# 3 - Prediction ==============================================================
# Predict conditional mean and standard deviation for each n_sim
## 3.1 - Produce tiles ---------------------------------------------------------
r <- covs[[1]]
# adjust the number of tiles according to the size and shape of your study area
t <- rast(nrows = 5, ncols = 5, extent = ext(r), crs = crs(r))
tile <- makeTiles(r, t,overwrite=TRUE,filename="02-Outputs/tiles/tiles.tif")

## 3.2 - Predict soil attributes per tiles -------------------------------------
# loop to predict on each tile for each simulation (n tiles x n simulation)
for (j in seq_along(tile)) {
  # where j is the number of tile
  for (i in 1:n_sim) {
    # where i is the number of simulation
    gc()
    # Read the tile j
    t <- rast(tile[j])
    # crop the covs with the tile extent
    covst <- crop(covs, t)
    # read the model i
    model <- readRDS(paste0("02-Outputs/models/model_",soilatt,"_", i,".rds"))
    # predict mean and sd
    meanFunc = function(x) mean(x, na.rm=TRUE)
    pred_mean <- terra::predict(covst, model = model$finalModel, na.rm = TRUE,
```

```
                                cpkgs="quantregForest", what=meanFunc)
    sdFunc = function(x) sd(x, na.rm=TRUE)
    pred_sd <- terra::predict(covst, model = model$finalModel, na.rm=TRUE,
                              cpkgs="quantregForest", what=sdFunc)
    # save the predicted tiles
    writeRaster(pred_mean,
                filename = paste0("02-Outputs/tiles/soilatt_tiles/",
                                  soilatt,"_tile_", j,"_model_",i,".tif"),
                overwrite = TRUE)
    writeRaster(pred_sd,
                filename = paste0("02-Outputs/tiles/soilatt_tiles/",
                                  soilatt,"_tileSD_", j,"_model_",i,".tif"),
                overwrite = TRUE)

    rm(pred_mean)
    rm(pred_sd)
    print(paste(tile[j],"model", i))
  }
}
```

Eventually, the predicted tiles are merged for each simulation (one for mean, one for standard deviation). Finally, two raster files are saved.

```
## 3.3 - Merge tiles both prediction and st.Dev for each simulation ------------
for (j in seq_along(tile)) {
  # merge tiles of simulation j
  f_mean <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                       pattern = paste0(soilatt,"_tile_",j,"_model_"),
                       full.names = TRUE)
  f_sd <- list.files(path = "02-Outputs/tiles/soilatt_tiles/",
                     pattern =  paste0(soilatt,"_tileSD_",j,"_model_"),
                     full.names = TRUE)
  # Estimate the mean of the n_sim predictions (both mean and sd) for each tile
  pred_mean <- rast(f_mean) %>% mean(na.rm=TRUE)
  pred_sd <- rast(f_sd) %>% mean(na.rm=TRUE)
  names(pred_sd) <- "sd"
  # save resulting tiles
  writeRaster(pred_sd,
```

```
              filename = paste0("02-Outputs/tiles/aggregated_tiles/",
                                soilatt,"_tilesd_", j,".tif"),
              overwrite = TRUE)
  writeRaster(pred_mean,
              filename = paste0("02-Outputs/tiles/aggregated_tiles/",
                                soilatt,"_tileMean_", j,".tif"),
              overwrite = TRUE)
  print(j)
}
```

In the following, the tiles are merged to one file that is outputted as raster file and saved in the maps folder.

```
## 3.4 - Merge tiles ------------------------------------------------------------
name_tiles <- c("sd", "Mean")
for (i in seq_along(name_tiles)) {
  # list tiles
  f <- list.files(path = "02-Outputs/tiles/aggregated_tiles/",
                  pattern = paste0(soilatt,"_tile",name_tiles[i]),
                  full.names = TRUE)
  # read tiles
  r <- list()
  for (g in seq_along(f)){
    r[[g]] <- rast(f[g])
    print(g)
  }
  # Mosaic tiles
  r <- sprc(r)
  r <- mosaic(r)
  # plot final map
  plot(r, main = paste("Predicted",name_tiles[i]))
  # save final map
  writeRaster(r, paste0("02-Outputs/maps/",soilatt,name_tiles[i],".tif"),
              overwrite=TRUE)
  rm(r)
  gc()
}
```

In case data with coordinates is available, the eval function is employed at the end to assess the gap between observed values with coordinates and the predicted ones without area-support.

```
# 4 - Validate resulting map ===================================================
# Load data with coordinates
val_data <- read_csv("01-Data/data_with_coord.csv") %>%
  vect(geom=c("x", "y"), crs = "epsg:4326")
# Load predited mean
pred_mean <- rast("02-Outputs/maps/p_brayMean.tif")
plot(pred_mean)
# extract values from predicted mean map
ex <- terra::extract(x = pred_mean, y = val_data, xy=F, ID=FALSE)
# Estimate accuracy indicators
val_data <- as.data.frame(val_data)
val_data <- cbind(val_data, ex)
val_data <- na.omit(val_data)

eval(val_data$mean,val_data[,soilatt])
```

At the end of the script, the files are again masked with the cropland mask layer and saved as raster files in the maps-output folder.

```
# 5 - Export final maps ========================================================
## 5.1 - Mask croplands ------------------------------------------------------
msk <- rast("01-Data/mask_arg.tif")
plot(msk)
# Mask croplands in predicted mean
pred_mean <- rast("02-Outputs/maps/p_brayMean.tif")
pred_mean <- mask(pred_mean, msk)
plot(pred_mean)
# Mask croplands in predicted sd
pred_sd <- rast("02-Outputs/maps/p_braysd.tif")
pred_sd <- mask(pred_sd, msk)
plot(pred_sd)

## 5.2 - Save results --------------------------------------------------------
writeRaster(pred_mean,
```

```
            paste0("02-Outputs/maps/",soilatt,"_no_coord.tif"),
            overwrite=TRUE)
writeRaster(pred_sd,
            paste0("02-Outputs/maps/",soilatt,"_no_coord_SD.tif"),
            overwrite=TRUE)
```

Annex IV: Quality assurance and quality control

The following protocol was devised to provide National Experts with a step-by-step guideline to perform a Quality Assurance (QA) and Quality Control (QC) of the 10 GSNmap first phase products.

The following protocol does not provide any guidance in terms of uncertainty estimation and validation. For more details and information on the estimation of uncertainties and potential map validation strategies please refer to Chapter 8.4.

Quality assurance and quality control consist of activities to ensure the quality of a particular result. Quality control is a reactive process that focuses on identifying defects and errors while quality assurance is a proactive approach aimed at preventing defects and errors. In the context of digital soil mapping, both processes are often interlinked. A QA is interlinked with a QC when it identifies defects and the QA remodels the process to eliminate the defects and prevent them from recurring (Chapman, 2005)(Figure10.2).

Each step in the following protocol should be considered in order to detect and eliminate errors, address data inaccuracies and assess the output completeness.

Step 1: Completeness of layers

The following Table **??**tab:products) gives an overview of all the GSNmap products in alphabetical order. Each product should include the ISO 3166-1

Figure 10.2: Quality assurance and quality control.

alpha-3 country code as uppercase letters in its name. For instance, in the case of Turkiye, ISO_GSNmap_Ntot_Map030 should be changed to TUR_GSNmap_Ntot_Map030.

All 10 soil property and soil nutrient maps with their corresponding 10 uncertainty layers must be georeferenced TIF (.tif) files.

Step 2: Check the projection and resolution of all data products

Open the products in QGIS or any other preferred GIS platform. Check that the projection of all products is EPSG:4326 - WGS 84 (Layer properties). Check that the spatial resolution (pixel size) (Layer properties) is equal to ~0.002246 degrees ; 250 m x 250 m at the equator.

Table 10.1: Data product overview.

Product	Filename
Major nutrients (3 files)	
Total Nitrogen map	ISO_GSNmap_Ntot_Map030.tiff
Available Phosphorus map	ISO_GSNmap_Pav_Map030.tiff
Total Potassium map	ISO_GSNmap_Ktot_Map030.tiff
Associated soil properties (7 files)	
Cation exchange capacity map	ISO_GSNmap_CEC_Map030.tiff
Soil pH map	ISO_GSNmap_pH_Map030.tiff
Soil clay map	ISO_GSNmap_Clay_Map030.tiff
Soil silt map	ISO_GSNmap_Silt_Map030.tiff
Soil sand map	ISO_GSNmap_Sand_Map030.tiff
Soil organic carbon map	ISO_GSNmap_SOC_Map030.tiff
Soil bulk density map	ISO_GSNmap_BD_Map030.tiff
Uncertainty maps (10 files)	
Total Nitrogen uncertainty map	ISO_GSNmap_Ntot_UncertaintyMap030.tiff
Available Phosphorus uncertainty map	ISO_GSNmap_Pav_UncertaintyMap030.tiff
Total Potassium uncertainty map	ISO_GSNmap_Ktot_UncertaintyMap030.tiff
Cation exchange capacity uncertainty map	ISO_GSNmap_CEC_UncertaintyMap030.tiff
Soil pH uncertainty map	ISO_GSNmap_pH_UncertaintyMap030.tiff
Soil clay uncertainty map	ISO_GSNmap_Clay_UncertaintyMap030.tiff
Soil silt uncertainty map	ISO_GSNmap_Silt_UncertaintyMap030.tiff
Soil sand uncertainty map	ISO_GSNmap_Sand_UncertaintyMap030.tiff
Soil organic carbon uncertainty map	ISO_GSNmap_SOC_UncertaintyMap030.tiff
Soil bulk density uncertainty map	ISO_GSNmap_BD_UncertaintyMap030.tiff

Step 3: Check the extent

Visualize the 20 products in QGIS or any preferred GIS platform. Load a land-use layer to visually assess that the simulations were done exclusively on croplands.

Step 4: Check the units, ranges, and outliers

In the following section possible value ranges for each product category (except available potassium) are presented. It is important to note that the provided ranges represent a gross approximation of the extremes within which the values should fall in. Results that fall outside these ranges need to be carefully evaluated based on local expertise and available literature.

The provided ranges can be compared in QGIS, R, or any preferred platform. Descriptive layer statistics can be viewed in QGIS under Layer Properties.

The following table (Table 10.2) presents ranges of possible values for 9 of the 10 mandatory GSNmap products. The ranges were calculated based on the distribution of the soil profile data within the World Soil Information Service (WoSIS), specifically the WoSIS snapshot 2019 (Batjes, N. H. *et al.* 2020). It is important to note that the data was not filtered for croplands and that the ranges were extracted from soil profiles sampled globally from a wide array of land covers and land uses.

QA/QC Script

The following script automates the for Steps described in the previous sections. It is important to note that the script's main objective is to provide a fast alternative to check the output layers and that it does not replace the need to visually assess the final maps based on expert knowledge.

```
#_____________________________________________________________________________
#
# QA/QC
# Soil Property Mapping
```

Table 10.2: Possible soil property and soil nutrient values based on the distribution of the values within the World Soil Information Service (WoSIS), specifically the WoSIS snapshot 2019.

Soil property	property_id	Unit	Min	1st Quartile	Median	3st Quartile
Total N	n_0_30	ppm	0.0	400.0	700.0	1500.0
P Bray I	p_0_30	ppm	0.0	1.6	5.0	16.0
P Mehlich 3	p_0_30	ppm	0.0	1.6	6.1	22.0
P Olsen	p_0_30	ppm	0.0	0.7	2.0	4.3
CEC	cec_0_30	cmol(c)/kg	0.1	7.5	14.0	23.0
pH (water)	ph_0_30	/	1.5	5.2	6.2	7.5
Clay	clay_0_30	%	0.0	11.1	21.9	35.3
Silt	silt_0_30	%	0.0	15.0	30.0	47.6
Sand	sand_0_30	%	0.0	15.0	36.0	60.2
Soil Organic Carbon	soc_0_30	%	0.0	2.0	5.1	14.0
Bulk density	bd_0_30	g/cm3	0.0	1.3	1.4	1.7
Available K	k_0_30	ppm	NA	NA	NA	NA

```
#
# GSP-Secretariat
# Contact: Isabel.Luotto@fao.org
#          Marcos.Angelini@fao.org
#-------------------------------------------------------------------------------

#Empty environment and cache
rm(list = ls())
gc()

# Content of this script ===============================================
# 0 - Set working directory and packages
# 1 - Step 1: Completeness of layers
# 2 - Step 2: Check the projection and resolution of all data products
# 3 - Step 3: Check the extent
# 4 - Step 4: Check the units, ranges, and outliers
#
# 5 - Export QA/QC report
```

```
#_______________________________________________________________________________


# 0 - Set working directory, soil attribute, and packages ======================

# Working directory
wd <- 'C:/Users/hp/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
#wd <- 'C:/Users/luottoi/Documents/GitHub/GSNmap-TM/Digital-Soil-Mapping'
setwd(wd)

# Define country of interes throuhg 3-digit ISO code
ISO ='ISO'

#load packages
library(terra)
library(readxl)

# Load reference values
dt <- read_xlsx("C:/Users/luottoi/Documents/GitHub/GSNmap-TM/tables/wosis_dist.xl
dt <- dt[!(dt$`Soil property` %in%c( "P Bray I","P Olsen" )),]

# In case old naming system was used
dt$old_prop_ids <- c('Ntot', 'Pav', 'CEC','pH', 'Clay', 'Silt', 'Sand', 'SOC', 'B


## Set potential ranges for Available K in ppm

dt[dt$property_id=='k_0_30','Min'] <- 0
dt[dt$property_id=='k_0_30','Max'] <- 150
# 1 - Step 1: Completeness of layers ------------------------------------------

#Check number of layers

## Specify number of soil property maps generated (not including the uncertainty

## Check if all layers were correctly generated (including uncertainty layers)
# and if the correct ISO code and soil property ids were included in the files na
files <- list.files(pattern= '.tif', full.names = T)
```

```
names <- list.files( pattern= '.tif', full.names = F)
names <- sub('.tif', '', names)


# Switch depending on the naming system (i.e. files have e.g. Pav instead of p_0_
#Step1 <-data.frame(property_id =dt$property_id)
Step1 <-data.frame(property_id =dt$old_prop_ids) #old naming system

Step1$Names <- 'Rename layer'
Step1$Uncertainty <- 'Missing'

for (i in unique(Step1$property_id)){

  t11 <- TRUE %in% grepl(paste0('SD_GSNmap_',i), files)|grepl(paste0('sd_',i), fi
  t12 <- TRUE %in% grepl(paste0(ISO,'_GSNmap_',i), files)|grepl(paste0('mean_',i)

  t13 <- TRUE %in% grepl(ISO, files)

  Step1[Step1$property_id ==i, 'Names'] <- ifelse(t12[[1]] ==T & t13[[1]] ==T, 'C
  Step1[Step1$property_id ==i, 'Uncertainty'] <- ifelse(t11[[1]] ==T , 'Generated

}


# 2 - Step 2: Check the projection and resolution of all data products ---------
r <- rast(files)
names(r) <- names
# Check projection (WGS 84)
(Step21=crs(r, describe=TRUE)$name =='WGS 84')

# Check resolution (250 m)
(Step22=round(res(r)[[1]], 5) == 0.00225)

# 3 - Step 3: Check the extent ------------------------------------------------
# Check if the layers were masked with a cropland mask

mask <- rast('mask/mask.tif')
mask <- project(mask, r[[1]])
```

```
t <- r[[1]]
t <- ifel(!is.na(t),1, NA)

t3 <- sum(values(t, na.rm=T))-sum(values(mask, na.rm=T))

(Step3= t3 <=10)



# 4 - Step 4: Check the units, ranges, and outliers ----------------------------
Step4 <- data.frame(property_id=Step1$property_id)
#Step4 <- data.frame(property_id =dt$property_id)
Step4$in_range <- 'Values not in range'

for (i in unique(Step4$property_id)){
  t41 <-min(values(r[[grepl(paste0('mean_',i), names(r))|grepl(paste0(ISO,'_GSNma
  t42 <-max(values(r[[grepl(paste0('mean_',i), names(r))|grepl(paste0(ISO,'_GSNma

  Step4[Step4$property_id ==i, 'in_range'] <- ifelse(t41[[1]] ==T & t42[[1]] ==T,

}



# 5 - Export QA/QC report ------------------------------------------------------
report <- merge(Step4, Step1, by=c('property_id'))

report$projection <- ifelse(Step21, 'WGS 84', 'Reproject layer')
report$resolution <- ifelse(Step21, '250 m', 'Resample layer')
report$extent <- ifelse(Step3, 'Croplands', 'Mask out layer')

report

write.csv(report, paste0('QA_QC_', ISO, '.csv'))
```

References

Arnon, D.I. & Stout, P. 1939. The essentiality of certain elements in minute quantity for plants with special reference to copper. *Plant physiology*, 14(2): 371.

Ballabio, C., Lugato, E., Fernández-Ugalde, O., Orgiazzi, A., Jones, A., Borrelli, P., Montanarella, L. & Panagos, P. 2019. Mapping LUCAS topsoil chemical properties at european scale using gaussian process regression. *Geoderma*, 355: 113912.

Beaudette, D.E., Roudier, P. & O'Geen, A. 2013. Algorithms for quantitative pedology: A toolkit for soil scientists. *Computers & Geosciences*, 52: 258–268.

Breiman, L. 2001. Random forests. *Machine Learning*, 45(1): 5–32. https://doi.org/10.1023/A:1010933404324

Brown, P.H., Zhao, F.-J. & Dobermann, A. 2022. What is a plant nutrient? Changing definitions to advance science and innovation in plant nutrition. *Plant and Soil*, 476(1): 11–23.

Cunningham, S.A., Attwood, S.J., Bawa, K.S., Benton, T.G., Broadhurst, L.M., Didham, R.K., McIntyre, S., Perfecto, I., Samways, M.J., Tscharntke, T. & others. 2013. To close the yield-gap while saving biodiversity will require multiple locally relevant strategies. *Agriculture, Ecosystems & Environment*, 173: 20–27.

Dokuchaev, V. 1883. *The russian chernozem report to the free economic society*. Imperial University of St. Petersburg: St. Petersburg)[in Russian].

Eisenstein, M. 2020. Natural solutions for agricultural productivity. *Nature*, 588: S59.

Fageria, N. & Knupp, A. 2014. Influence of lime and gypsum on growth and yield of upland rice and changes in soil chemical properties. *Journal of Plant Nutrition*, 37(8): 1157–1170.

FAO. 2022. *Soils for nutrition: State of the art.* FAO.
FAO, U., IFAD & WHO. 2022. *The state of food security and nutrition in the world 2022.* FAO.
Hebebrand, C. & Laborde, D. 2022. High fertilizer prices contribute to rising global food security concerns. https://www.ifpri.org/blog/high-fertilizer-prices-contribute-rising-global-food-security-concerns.
Hengl, T., Leenaars, J.G.B., Shepherd, K.D., Walsh, M.G., Heuvelink, G.B.M., Mamo, T., Tilahun, H., Berkhout, E., Cooper, M., Fegraus, E., Wheeler, I. & Kwabena, N.A. 2017. Soil nutrient maps of sub-saharan africa: Assessment of soil nutrient content at 250 m spatial resolution using machine learning, 109: 77–102. https://doi.org/10.1007/s10705-017-9870-x
Hengl, T., Miller, M.A., Križan, J., Shepherd, K.D., Sila, A., Kilibarda, M., Antonijević, O., Glušica, L., Dobermann, A., Haefele, S.M. & others. 2021. African soil properties and nutrients mapped at 30 m spatial resolution using two-scale ensemble machine learning. *Scientific Reports*, 11(1): 1–18.
Janssen, P.H.M. & Heuberger, P.S.C. 1995. Calibration of process-oriented models, 83: 55–66. https://doi.org/10.1016/0304-3800(95)00084-9
Jenny, H. 1941. A system of quantitative pedology. In "factors of soil formation". McGraw Hill: New York.
Khaledian, Y. & Miller, B.A. 2020. Selecting appropriate machine learning methods for digital soil mapping. *Applied Mathematical Modelling*, 81: 401–418.
Kuhn, M. 2022. *Caret: Classification and regression training.* (also available at https://CRAN.R-project.org/package=caret).
Kursa, M.B. & Rudnicki, W.R. 2010. Feature selection with the Boruta package. *Journal of Statistical Software*, 36(11): 1–13. (also available at https://doi.org/10.18637/jss.v036.i11).
Lamichhane, S., Kumar, L. & Wilson, B. 2019. Digital soil mapping algorithms and covariates for soil organic carbon mapping and their implications: A review. *Geoderma*, 352: 395–413.
Liebig, J. von. 1841. *Die organische chemie in ihrer anwendung auf agricultur und physiologie.* Vieweg.
Ma, Y., Minasny, B., Malone, B.P. & Mcbratney, A.B. 2019. Pedology and digital soil mapping (DSM). *European Journal of Soil Science*, 70(2): 216–235.
McBratney, A.B., Santos, M.L.M. & Minasny, B. 2003. On digital soil mapping, 117: 3–52. https://doi.org/10.1016/s0016-7061(03)00223-4
Meinshausen, Ni. 2006. Quantile regression forests. *Journal of Machine*

Learning Research, 7(6).
Mengel, K. & Kirkby, E.A. 2012. *Principles of plant nutrition*. Springer Science & Business Media.
Mengel, K., Kirkby, E.A., Kosegarten, H. & Appel, T. 2001. Plant nutrients. *Principles of plant nutrition*, pp. 1–13. Springer.
Olmedo, G., Rodriguez, D. & Angelini, M. 2017. Advances in digital soil mapping and soil information systems in argentina. *GlobalSoilMap*, pp. 13–16. CRC Press.
Padarian, J., Minasny, B. & McBratney, A.B. 2019. Using deep learning for digital soil mapping. *Soil*, 5(1): 79–89.
Robertson, G.P., Sollins, P., Ellis, B.G., Lajtha, K. & others. 1999. Exchangeable ions, pH, and cation exchange capacity. *Standard soil methods for long-term ecological research*, 2: 462.
Sainz Rozas, H.R., Eyherabide, M., Larrea, G.E., Martinez Cuesta, N., Angelini, H.P., Reussi Calvo, N.I. & Wyngaard, N. 2019. Relevamiento y determinación de propiedades químicas en suelos de aptitud agrícola de la región pampeana. *Fertilizar, Argentina*, 8(9): 12.
Van Sundert, K., Radujković, D., Cools, N., De Vos, B., Etzold, S., Fernández-Martínez, M., Janssens, I.A., Merilä, P., Peñuelas, J., Sardans, J. & others. 2020. Towards comparable assessment of the soil nutrient status across scales—review and development of nutrient metrics. *Global change biology*, 26(2): 392–409.
Vicca, S., Stocker, B.D., Reed, S., Wieder, W.R., Bahn, M., Fay, P.A., Janssens, I.A., Lambers, H., Peñuelas, J., Piao, S. & others. 2018. Using research networks to create the comprehensive datasets needed to assess nutrient availability as a key determinant of terrestrial carbon cycling. *Environmental Research Letters*, 13(12): 125006.
Wadoux, A.M.-C., Minasny, B. & McBratney, A.B. 2020. Machine learning for digital soil mapping: Applications, challenges and suggested solutions. *Earth-Science Reviews*, 210: 103359.
Wadoux, A.M.J.-C., Heuvelink, G.B.M., Bruin, S. de & Brus, D.J. 2021. Spatial cross-validation is not the right way to evaluate map accuracy, 457: 109692. https://doi.org/10.1016/j.ecolmodel.2021.109692
Wadoux, A.M.J.-C., Walvoort, D.J.J. & Brus, D.J. 2022. An integrated approach for the evaluation of quantitative soil maps through taylor and solar diagrams, 405: 115332. https://doi.org/10.1016/j.geoderma.2021.115332
Wang, D., Marschner, P., Solaiman, Z. & Rengel, Z. 2007. Growth, p uptake and rhizosphere properties of intercropped wheat and chickpea in soil amended with iron phosphate or phytate. *Soil Biology and Biochemistry*, 39(1):

249–256.
Wright, M.N., Ziegler, A. & König, I.R. 2016. Do little interactions get lost in dark random forests? *BMC bioinformatics*, 17(1): 1–10.
Yigini, Y., Olmedo, G., Reiter, S., Baritz, R., Viatkin, K. & Vargas, R. 2018. Soil organic carbon mapping: cookbook

GLOBAL SOIL PARTNERSHIP

The Global Soil Partnership (GSP) is a globally recognized mechanism established in 2012. Our mission is to position soils in the Global Agenda through collective action. Our key objectives are to promote Sustainable Soil Management (SSM) and improve soil governance to guarantee healthy and productive soils, and support the provision of essential ecosystem services towards food security and improved nutrition, climate change adaptation and mitigation, and sustainable development.

Australian Government
Department of Agriculture, Water and the Environment

AFACI
Asian Food & Agriculture Cooperation Initiative

Ministry of Finance of the Russian Federation

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Rural Development Administration