EIS
Exploration Information System

D 3.5: EIS Toolkit Final Release

User Manual and Technical Specifications

Version 1.0

Lead Beneficiary: GTK
04 / 2024

Johanna Torppa[1], Bijal Chudasama[1], Pyry Lehtonen[1], Nikolas Ovaskainen[1], Niko Aarnio[2], Timo Aarnio[2], Miikka Kallio[2], Mika Sorvoja[2], Ina Storch[3], Peggy Hielscher[3], Michael Steffen[3], Andreas Knobloch[3], Fahimeh Farahnakian [1,4], Dipak Nidhi [4], Luca Zelioli [4]

[1]*GTK*
[2]*GISPO*
[3]*Beak Consultants GmbH*
[4]*University of Turku*

Funded by
the European Union

Disclaimer

Funded by
the European Union

Document information

Grant Agreement / Proposal ID	101057357
Project Title	Exploration Information System
Project Acronym	EIS
Scientific Coordinator	Vesa Nykänen (vesa.nykanen@gtk.fi) – GTK
Project starting date (duration)	1 May 2022 (36 months)
Related Work Package	WP 3
Related Task(s)	Task 3.3
Lead Organisation	GTK
Contributing Partner(s)	GISPO, BEAK, UTU
Due Date	30.04.2024
Submission Date	02.05.2024
Dissemination level	PU

History

Date	Version	Submitted by	Reviewed by	Comments
01.02.2024	0.1	Andreas Knobloch		Template
30.04.2024	0.2	Johanna Pesonen		Draft
02.05.2024	0.9	Andreas Knobloch		Revision
02.05.2024	1.0	Hafsa Munia		Submission

Funded by
the European Union

Table of contents

Funded by
the European Union

Abbreviations and Acronyms

Acronym	Description
WP	Work Package
GIS	Geographic Information System
EIS	Exploration Information System
GUI	Graphical User Interface

Summary

The Deliverable D3.5 provides a description of the contents and general structure of the EIS Toolkit final release. As more versions are likely to be released this release will be referred to "first stable release" later in this document.

Keywords

Software Design, Mineral Prospectivity Modelling, Mineral Predictive Mapping, QGIS, Artificial Intelligence, Toolkit, Wizard, Python

Funded by
the European Union

1. Executive summary

This document describes the first stable release of the “EIS Toolkit”, which is a standalone Python library for conducting Mineral Prospectivity Mapping. The toolkit utilizes industry-standard libraries for well-defined tasks and provides an interface for integrations both with the upcoming “QGIS EIS Plugin” and other software.

A common way to use this kind of a library is via Jupyter Notebooks, which is an interactive environment for running Python software and a sample of such use is included with this deliverable in Appendix 1 both as a runnable/interactive notebook file and a static export that visualizes the workflow and its results. The interactive version works as a user manual for end users.

The technical documentation for “EIS Toolkit” is automatically generated and is described in chapter 4 and an export of the documentation is included with this deliverable in Appendix 2.

This and future releases and related documentation can be downloaded from “EIS Toolkit” GitHub repository which is located at: https://github.com/GispoCoding/eis_toolkit.

Funded by
the European Union

2. Introduction

2.1 General remarks

This document provides background information about the first stable release of "EIS Toolkit".

The documentation for EIS Toolkit is currently in two forms:

- a Jupyter Notebook that guides the user through an example workflow (User Guide) (see Appendix 1) and
- a Technical Specification document that includes documentation for the actual EIS Toolkit functionality (see Appendix 2).

2.2 EIS WP3 and Task 3.3

The main objective of WP3 is the development of a GIS (Geographical Information System)-based Exploration Information System (EIS) for predictive mapping of mineral resources. EIS does not have a strict definition but can be characterized as an environment for performing data analysis and modelling, for managing data and other information, and for representing results in various forms.

In Task 3.3, a library of Python functions for EIS is implemented. We call this library the "EIS Toolkit", and it is a comprehensive collection of independent functions relevant for performing mineral prospectivity analysis related tasks. These tasks include mainly predictive mapping and data integration via mathematical modelling, but also some general data processing and analysis. Also, tools for evaluation of the goodness of the models and modelling results are included for efficient decision making in the identification and prioritization of exploration targets.

The "EIS Toolkit" contains implementations of existing and new algorithms, and new functions can be added even after the EIS project. Emphasis is given to exploring the applicability of modern machine learning methods, such as convolutional neural networks, the use of which in mineral prospectivity modelling is still at its infancy.

The structure of and interfaces to the functions included in the library have been planned so that the functions can be smoothly integrated to other software, such as the "EIS QGIS Plugin". Existing Python libraries have been reviewed and are used to minimize the coding effort.

Funded by
the European Union

3. User manual

This chapter outlines how to use "EIS Toolkit", what are the system requirements, how it can be installed and a Jupyter Notebook for interactive use with guidance.

Additionally, a static export of the Jupyter Notebook is provided in **Appendix 1** that visualizes an example workflow when done inside Jupyter Notebook

3.1 System requirements

The only system requirement for installing the "EIS Toolkit" through the Python package is to have a supported version of Python installed on the system. The "EIS Toolkit" supports all Python 3.9.x and 3.10.x versions. The compatible Python version should be either a system Python installation or in an environment management system, such as Conda.

3.2 Installation guide

The two primary ways of installing EIS Toolkit are installation to a Conda environment from the conda-forge channel and installation to Python venv from PyPI (Python Packaging Index). These are two widely used online repositories that host various Python packages and enable easy download and installation into Python environments.

To install EIS Toolkit in a Conda environment, the user should create a new, empty Conda environment that uses either Python 3.9 or Python 3.10. Then, when inside the environment, the installation is done with the command

conda install –c conda-forge eis_toolkit

To install EIS Toolkit in a venv, the user should create similarly a fresh environment and use the command

pip install eis_toolkit

The package automatically installs all the necessary Python libraries, and EIS Toolkit will be ready to use. Sometimes, for some platforms, a package called GDAL might cause installation issues. This is not a fault of EIS Toolkit, but guidance to overcome related issues are given in the GitHub repository, namely acquiring and installing a compatible GDAL binary separately before attempting to install EIS Toolkit.

3.3 Jupyter Notebook description

Jupyter notebooks are a widely used format to run Python script blocks with lasting outputs. They can be especially useful for data analysis work, which is why the format is quite popular in academia.

During the development of EIS Toolkit, Jupyter notebooks were utilized to demonstrate the developed tools. These notebooks are collected in the development repository of EIS Toolkit in GitHub, in folder called notebooks directly under the root folder. Even after the main phase of development, they serve as one sort of documentation, even if not the primary one.

Funded by
the European Union

In addition to the notebooks that demonstrate individual tools, a workflow demonstration notebook was crafted to show more comprehensively how a MPM workflow with EIS Toolkit might look like. An export of this notebook is provided in this deliverable.

4. Technical specifications

The "EIS Toolkit" encompasses a comprehensive suite of functions designed to facilitate efficient and effective mineral exploration. These functions are meticulously organized into distinct modules, each serving a specific purpose within the exploration workflow. Technical specifications for the "EIS Toolkit" functions are automatically generated from docstrings, which are extensively documented within the source code. Below is an overview of the key modules.

For further details on the technical specifications and functionalities of each module, please refer to **Appendix 2**.

In the following sections, the numbers in brackets refer to the chapters / section numbers in the Technical Specifications in Appendix 2.

4.1 Conversions module

The Conversions Module in the "EIS Toolkit" provides essential functionalities for converting different types of geospatial data, facilitating seamless integration and analysis within the exploration workflow.

CSV to Geo Data Frame Conversion (4.1.1):

This function enables users to read CSV files containing geospatial data and convert them into Geo Data Frames. It supports the transformation of point data represented by X and Y coordinates or geometric shapes specified in Well-Known Text (WKT) format. Users can also define the target Coordinate Reference System (CRS) for the resulting Geo Data Frame.

Raster to Data Frame Conversion (4.2.2):

The Raster to Data Frame conversion function allows users to convert raster datasets into Pandas Data Frames. Users can select specific bands from multi-band raster or utilize all bands for conversion. Additionally, pixel coordinates (row, col) can be optionally included in the Data Frame for each pixel of the raster, providing valuable spatial context to the data.

By utilizing these conversion functionalities, users can seamlessly transform and prepare diverse geospatial datasets for further analysis and exploration within the "EIS Toolkit."

4.2 Evaluation module

The Evaluation Module in the "EIS Toolkit" provides essential tools for evaluating the performance and reliability of predictive models and data analyses.

Calculate Base Metrics (4.2.1):
The Calculate Base Metrics function computes true positive rate, proportion of area, and false positive rate values for different thresholds. It leverages mineral deposit locations and mineral prospectivity maps to evaluate model performance comprehensively.

Evaluate Classification Labels (4.2.2):
This module includes functions that together generate a comprehensive report of classification results. Accuracy, precision, recall F1-score and confusion matrix are produced for the given test results.

Evaluate Classification Probabilities (4.2.3):
This module includes several tools that together form a comprehensive and advanced set to evaluate probability array output from a classifier. The user can calculate metrics, such as ROC AUC, log loss, average precision, and Brier score loss. In addition, they can plot several plots including ROC curve, DET curve, precision-recall curve and the calibration curve. This module supports direct evaluation of classification labels by giving information that can be used to derive custom classification threshold values or directly assess goodness of a classifier model.

Plot Confusion Matrix (4.2.4):
The Plot Confusion Matrix function generates a heatmap visualization of the confusion matrix. This tool visualizes the binary classification results in a clear way, indicating the true/false negatives/positives with color, percentages and counts.

Plot Neural Network Model Performance (4.2.5):
The Plot Correlation Matrix function generates a heatmap visualization of the correlation matrix. This visualization aids in understanding the relationships between variables, providing insights into potential multicollinearity and the overall structure of the data.

Plot Prediction-Area (P-A) Curves (4.2.6):
This function plots prediction-area (P-A) curves, which are valuable for evaluating mineral prospectivity maps and evidential layers. By visualizing true positive rate and proportion of area values against threshold values, users can assess the performance of predictive models.

Plot Rate Curve (4.2.7):
The Plot Rate Curve function generates success rate, prediction rate, or ROC curves based on provided X and Y values. This visualization helps users understand the trade-offs between true positive rate and false positive rate, aiding in model selection and evaluation.

Score Model (4.2.8):
This tools allow for direct and simple evaluation of a classifier or regressor model by scoring it with the user-defined metric. The supported metrics are accuracy, recall, precision, F1, MAE, MSE, RMSE and R2.

4.3 Exploratory analyses module

The Exploratory Analyses Module within the "EIS Toolkit" encompasses a diverse range of functions aimed at uncovering insights and relationships within geospatial data, facilitating informed decision-making in mineral exploration endeavors.

Funded by
the European Union

Basic Exploratory Plots (4.3.1):
This module groups basic plot types that can be used to gain insight on data. These plots include histogram plot, KDE (kernel density estimate) plot, ECDF (estimator of cumulative distribution function) plot, line plot, scatterplot, heatmap, pair plot, regression plot, bar plot and box plot.

Chi-square Test (4.3.2):
The Chi-square test assesses independence between categorical variables.

Plot Correlation Matrix (4.3.3):
The Plot Correlation Matrix function generates a heatmap visualization of the correlation matrix. This visualization aids in understanding the relationships between variables, providing insights into potential multicollinearity and the overall structure of the data.

Plot Covariance Matrix (4.3.4):
The Plot Covariance Matrix function generates a heatmap visualization of the covariance matrix. This visualization works similarly as Plot Correlation Matrix, but for covariance matrix data.

DBSCAN Clustering (4.3.5):
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is utilized to identify clusters within geospatial data based on density. By specifying parameters such as maximum distance and minimum samples, users can effectively identify spatial patterns indicative of mineralization occurrences.

Descriptive Statistics (4.3.6):
This function computes descriptive statistics from both vector and raster data, providing insights into the distribution and variability of geospatial attributes essential for exploration analysis and decision-making.

Feature Importance Evaluation (4.3.7):
Evaluate the importance of features derived from geospatial data using machine learning models. By assessing the contribution of each feature to model performance, users can prioritize exploration targets effectively.

K-means Clustering (4.3.8):
Perform K-means clustering on geospatial data to partition it into distinct clusters. This aids in identifying spatially coherent regions or anomalies associated with mineralization occurrences.

Local Moran's I (4.3.9):
This tool calculates the Local Moran's I statistic that assess local spatial autocorrelation of data. Using this tool, the user can identify anomalous clusters or other spatial patterns in their data which can be crucial to be aware of.

Test Normality (4.3.10):
This tool tests whether the given data sample is likely from a normally distributed population. Knowing if data is normally distributed can be important when running certain tools and understanding the nature of data at hand.

Plot Parallel Coordinates (4.3.11):
Generate parallel coordinates plots to visualize multivariate relationships within geospatial datasets. This aids in identifying patterns and trends across multiple variables, enhancing understanding of mineralization indicators and exploration targets.

Principal Component Analysis (PCA) (4.3.12):
Conduct PCA to reduce the dimensionality of geospatial datasets while preserving essential information. Visualization of principal components assists in identifying spatial patterns and relationships crucial for mineral exploration.

Funded by
the European Union

Through the utilization of these exploratory analyses functions, users can gain valuable insights into the spatial characteristics of geological features and mineral occurrences, facilitating more informed and effective decision-making in mineral exploration endeavors.

4.4 Prediction module

The Prediction Module in the "EIS Toolkit" equips users with powerful tools for predicting mineral occurrences and evaluating prospectivity.

Fuzzy Overlay Operations (4.4.1):
This suite of functions enables users to perform various fuzzy overlay operations on raster data. By utilizing fuzzy logic techniques such as AND, OR, product, sum, and gamma overlay, users can analyze multi-dimensional geospatial data to identify areas of interest and assess mineral prospectivity.

Weights of Evidence Analysis (4.4.2):
These functions facilitate the calculation of weights of evidence, allowing users to assess the spatial associations between input data and mineral deposits. By determining posterior probabilities and weights of spatial association, users can make informed decisions regarding mineral exploration targets and planning.

Random Forest (4.4.3):
TBD

Logistic Regression (4.4.4):
TBD

Gradient Boosting (4.4.5):
TBD

Multilayer Perceptron (4.4.6):
TBD

4.5 Raster processing module

The Raster Processing module offers a comprehensive suite of functions for efficiently manipulating raster data, ensuring its compatibility and suitability for various analytical tasks. These tools enable users to address diverse raster data processing needs, including data validation, spatial analysis, and format standardization. By providing a robust set of functionalities, the module empowers users to seamlessly manage and optimize raster data for subsequent analysis and modeling tasks, enhancing the efficiency and accuracy of geospatial workflows.

Clipping (4.5.1):
Clips a raster with polygon geometries, extracting specific regions of interest.

Funded by
the European Union

Create Constant Raster (4.5.2):
Generates a constant raster based on user-defined parameters, offering flexibility in defining extent and coordinate system.

Distance to Anomaly (4.5.3):
Computes a new raster where each pixel value is the euclidean distance to nearest anomaly pixel. The user can define the anomaly threshold and criteria (higher, lower, in-between, outside). This tool is important in processing mineral proxies and preprocessing datasets that represent anomalies.

Extract Values from Raster (4.5.4):
Extracts raster values using point data to a Data Frame, facilitating data extraction for analysis.

Filters (4.5.5):
The filters module contains a comprehensive set of different filter functions that users can apply to process their raster data.

Reclassify (4.5.6):
The reclassify module implements the most commonly used reclassification methods, namely manual breaks, defined intervals, equal intervals, quantiles, natural breaks, geometrical intervals and standard deviation. These reclassification methods are used to discretize raster data, which is a necessary preprocessing step for Weights of Evidence modeling method.

Reprojecting (4.5.7):
Re-projects a raster to match a given coordinate reference system (CRS), ensuring compatibility between datasets.

Resampling (4.5.8):
Resamples a raster according to a specified resolution, maintaining data integrity during transformation.

Snapping (4.5.9):
Aligns a raster to a reference grid raster, ensuring alignment for consistent analysis.

Surface Derivatives (4.5.10):
The surface derivatives module includes both first and second order surface derivatives to offer an extensive amount of surface attributes. These surface attributes can be processed to be used in models in certain contexts.

Unifying (4.5.11):
Unifies given raster relative to a base raster by re-projecting, resampling, aligning, and optionally clipping them. This operation ensures consistency in grid properties and extents across multiple raster.

Unique Combinations (4.5.12):
Produces a raster where each pixel value marks a certain unique combination between input rasters in the given location.

Windowing (4.5.13):
Extracts a window from a raster centered at specified coordinates, allowing for localized analysis. Padding with no data values is applied if the window extends beyond the raster bounds.

Funded by
the European Union

4.6 Training data tools module

The Training Data Tools module equips users with essential functionalities for preparing training datasets, particularly in the context of machine learning tasks. By offering these tools, the module facilitates the creation of balanced and well-prepared training datasets, crucial for achieving optimal model performance and accuracy in machine learning applications.

Class Balancing (4.6.1):
With this function users can address class imbalance issues by applying the SMOTETomek resampling method, ensuring a more representative distribution of classes in the training data. It adjusts the class distribution by oversampling the minority class and under-sampling the majority class to mitigate class imbalance issues. Users can specify various parameters such as the sampling strategy and random state to customize the resampling process. The function returns the resampled feature matrix and target labels, ensuring a more balanced dataset for training machine learning models.

4.7 Transformations module

The Transformations module provides a suite of functions for preprocessing raster data, enabling users to apply various transformations to prepare their data for modeling and analysis. These functions include binarization, clipping, normalization, logarithmic transformation, sigmoid transformation, and winsorization, offering flexibility in data preprocessing to suit different modeling needs. Whether it's transforming data into binary format, clipping values based on specific thresholds, or normalizing data to a standard scale, these tools empower users to preprocess raster data effectively before further analysis or modeling tasks.

Binarize (4.7.1):
This function binarizes raster data based on a given threshold. It replaces values less than or equal to the threshold with 0 and values greater than the threshold with 1.

Clip (4.7.2):
The clip transform function clips raster data based on specified upper and lower limits. It replaces values below the lower limit and above the upper limit with provided values.

CoDa Transformations (4.7.3):
TBD

Linear (4.7.4):
Min-max-scaling normalizes raster data based on a specified new range. It transforms the data into the new interval defined by the provided minimum and maximum values.

Z-Score Normalization (4.7.5):
This function normalizes raster data based on the mean and standard deviation. The resulting data will have a mean of 0 and a standard deviation of 1.

Logarithmic (4.7.6):
The log-transform function performs a logarithmic transformation on the provided raster data. It replaces

Funded by
the European Union

negative values with a specified no data value and applies the logarithmic transformation based on the chosen base.

One-hot Encode (4.7.7):
The one-hot encode tool creates binary attributes to data from a multi-value attribute. To run deep learning models such as MLP, having this tool available is necessary.

Sigmoid (4.7.8):
The sigmoid-transform transforms data into a sigmoid shape based on a specified new range. It uses parameters such as minimum, maximum, slope, and center to perform the transformation.

Winsorize (4.7.9):
The winsorize function winsorizes raster data based on specified percentile values. It replaces values outside the specified percentile ranges with the nearest values within the range.

4.8 Vector processing module

The Vector Processing Module in the "EIS Toolkit" offers a range of functionalities for handling and analyzing vector data, facilitating geospatial analysis tasks.

Calculate Geometry (4.8.1):
Calculates area for each polygon feature and length for each line feature in the input vector data.

Cell-Based Association (4.8.2):
This function initializes a Cell-Based Association (CBA) matrix from vector files. It calculates the mesh based on geometries contained in the file and cell size specified by the user. Users can add multiple vector datasets to the matrix, incorporating targeted shapes and attributes for comprehensive analysis.

Distance Computation (4.8.3):
The Distance Computation function calculates the distance from raster cells to the nearest geometry in a given set of vector data. This tool is valuable for proximity analysis and understanding spatial relationships between raster and vector datasets.

Extract Shared Lines (4.8.4):
Extracts shared lines from an input vector dataset.

IDW Interpolation (4.8.5):
The Inverse Distance Weighted (IDW) interpolation function performs spatial interpolation on vector data, generating a raster output. Users can specify the target column containing values for interpolation, resolution of the output raster, and the power parameter to control the rate at which weights decrease with distance.

Kriging Interpolation (4.8.6):
This function implements Kriging interpolation on input vector data, producing a raster output with interpolated values. Users can select the variogram model, coordinates type, and method (ordinary or universal) for Kriging interpolation, providing flexibility in spatial analysis tasks.

Rasterize Vector (4.8.7):
Transforming vector data into raster format is facilitated by the Rasterize Vector function. Users can specify

Funded by
the European Union

parameters such as resolution, value column, default value, and merge strategy to customize the rasterization process according to their analysis requirements.

Reproject Vector (4.8.8):
Re-projecting vector data to match a given coordinate reference system (CRS) is achieved with the Re-project Vector function. Users can specify the target CRS using its EPSG code, ensuring consistency and compatibility across geospatial datasets.

Vector Density (4.8.9):
The Vector Density function computes the density of geometries within a raster grid. Users can define parameters such as resolution, base raster profile, buffer value, and statistic (e.g., density).

Funded by
the European Union

5. Conclusion

The first stable release of the “EIS Toolkit” has been completed and published at the GitHub page. It has been internally tested among WP3 partners and is ready to be utilized and applied through the “EIS QGIS Plugin” by EIS WP4 partners in the EIS test sites.

The implemented tools from the EIS Toolkit in the EIS QGIS Plugin are being described in Deliverable D3.6, which is being submitted simultaneously.

Development in EIS project under WP3 has now being completed in Task 3.2 (Additional Algorithms) and Task 3.3 (actual EIS Toolkit development), and will fully shift now to Task 3.5 to further develop the beta release of the “EIS QGIS Plugin”, whose first stable release is aimed for October 2024.

Based on feedback from WP4 partners in EIS using the EIS QGIS Plugin, potential bug-fixes an improvements of the implemented tools from the EIS Toolkit will continue though in WP3 until the end of the EIS project duration.

Funded by
the European Union

6. References

Nykänen, V., Lahti, I., Niiranen, T., & Korhonen, K. (2015). Receiver operating characteristics (ROC) as validation tool for prospectivity models — A magmatic Ni–Cu case study from the Central Lapland Greenstone Belt, Northern Finland. Ore Geology Reviews, 71, 853-860. doi:10.1016/j.oregeorev.2014.09.007

Funded by
the European Union

Appendix 1: EIS Toolkit – Workflow Demonstration

Funded by
the European Union

EIS Toolkit workflow demo

April 28, 2024

This Jupyter notebook serves as a first user guide to beta testers of EIS Toolkit. It shows how to import and call various tools of EIS Toolkit and includes steps of a simple MPM workflow.

Note that to run this notebook without modifications you need to have the test data (and under correct folder structure). One can always change the filepaths and use their own data, granted that the data is in right the format and is meaningful geospatial data. However, the primary function of this notebook is not to provide a template for conducting MPM workflows, but instead serve as example and showcase some of the tools of EIS Toolkit.

0.0.1 1. Imports and filepath definitions

[ ]:
```
import os
import rasterio
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt

from rasterio.io import MemoryFile
from rasterio.plot import show

import sys
sys.path.insert(0, "..")

from eis_toolkit.exploratory_analyses.basic_plots_seaborn import pairplot,␣
 ↪kdeplot
from eis_toolkit.exploratory_analyses.parallel_coordinates import␣
 ↪plot_parallel_coordinates
from eis_toolkit.exploratory_analyses.descriptive_statistics import␣
 ↪descriptive_statistics_raster
from eis_toolkit.exploratory_analyses.pca import compute_pca

from eis_toolkit.prediction.fuzzy_overlay import gamma_overlay
from eis_toolkit.prediction.weights_of_evidence import␣
 ↪weights_of_evidence_calculate_weights,␣
 ↪weights_of_evidence_calculate_responses

from eis_toolkit.raster_processing.unifying import unify_raster_grids
from eis_toolkit.raster_processing.distance_to_anomaly import␣
  ↪distance_to_anomaly

from eis_toolkit.transformations.sigmoid import _sigmoid_transform
from eis_toolkit.transformations.linear import _min_max_scaling

from eis_toolkit.vector_processing.idw_interpolation import idw
from eis_toolkit.vector_processing.distance_computation import␣
  ↪distance_computation

from eis_toolkit.evaluation.calculate_base_metrics import calculate_base_metrics
from eis_toolkit.evaluation.plot_rate_curve import plot_rate_curve

from eis_toolkit.utilities.nodata import set_raster_nodata
```

[ ]:
```
# Folder with the test data in it. Modify this match the location of your test␣
  ↪data folder.
test_data_folder = "../tests/data/local/workflow_demo"

AEM_inphase_fp = os.path.join(test_data_folder, "IOCG_AEM_Inph_.tif")
AEM_quad_fp = os.path.join(test_data_folder, "IOCG_AEM_Quad.tif")
AEM_ratio_fp = os.path.join(test_data_folder, "IOCG_EM_ratio.tif")
Magn_AS_fp = os.path.join(test_data_folder, "IOCG_Magnetic.tif")

till_geochem_fp = os.path.join(test_data_folder,␣
  ↪"IOCG_CLB_Till_Geochem_reg_511p.shp")
structures_fp = os.path.join(test_data_folder, "IOCG_CLB_Structures_1M.shp")
lithology_fp = os.path.join(test_data_folder, "IOCG_CLB_Lith_Asstn_1M.shp")
known_occurances_fp = os.path.join(test_data_folder, "IOCG_Deps_Prosp_Occs.shp")
```

[ ]:
```
# Additionally, we suppress warnings, because the possible warnings are not␣
  ↪related to EIS Toolkit but Seaborn
# and other underlaying libraries.
import warnings
warnings.filterwarnings('ignore')
```

0.1 2. Preprocess data

For preprocessing, three main tools to create proxy data for modeling are showcased: Interpolation of vector data (typically geochemical data), computing distances to vector features (typically geological data) and computing distances to anomalous pixels (typically geophysical data). Beside these, some other tools and steps to prepare data are used.

Selecting a base raster As the first thing, one raster should be selected to be the base raster with desired grid properties. This raster is read and its profile saved to be used in processing tools.

```
[ ]: # Select AEM_inphase raster profile as the baes raster
     with rasterio.open(AEM_inphase_fp) as AEM_inphase:
         raster_profile = AEM_inphase.profile
```

Preprocess geochemical data Interpolate selected element concentrations.

```
[ ]: till_geochem = gpd.read_file(till_geochem_fp)
```

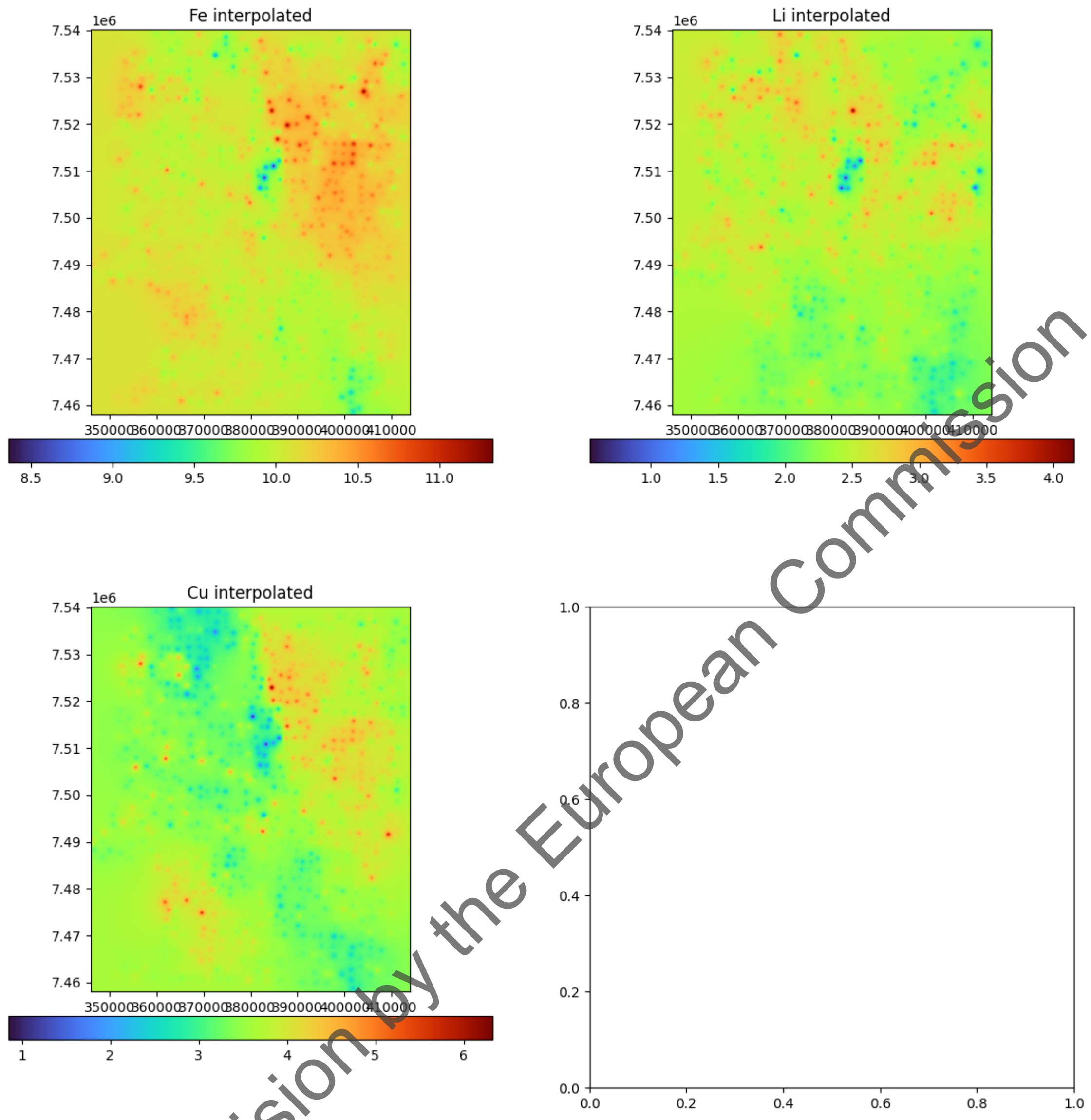
```
[ ]: # Iron interpolated
     till_geochem["fe_log"] = np.log(till_geochem["Fe_ppm_511"])
     fe_interpolated = idw(geodataframe=till_geochem, target_column="fe_log",␣
      ↪raster_profile=raster_profile, power=2)
```

```
[ ]: # Lithium interpolated
     till_geochem["li_log"] = np.log(till_geochem["Li_ppm_511"])
     li_interpolated = idw(till_geochem, "li_log", raster_profile)
```

```
[ ]: # Copper interpolated
     till_geochem["cu_log"] = np.log(till_geochem["Cu_ppm_511"])
     cu_interpolated = idw(till_geochem, "cu_log", raster_profile)
```

```
[ ]: # Visualize interpolated element concentrations
     fig, axs = plt.subplots(2, 2, figsize = (14, 14))
     cmap = plt.get_cmap('turbo')

     axs[0, 0].set_title("Fe interpolated")
     clrbar = axs[0, 0].imshow(fe_interpolated, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(fe_interpolated, ax = axs[0, 0], transform = raster_profile["transform"],␣
      ↪cmap=cmap)

     axs[0, 1].set_title("Li interpolated")
     clrbar = axs[0, 1].imshow(li_interpolated, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(li_interpolated, ax = axs[0, 1], transform = raster_profile["transform"],␣
      ↪cmap=cmap)

     axs[1, 0].set_title("Cu interpolated")
     clrbar = axs[1, 0].imshow(cu_interpolated, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(cu_interpolated, ax = axs[1, 0], transform = raster_profile["transform"],␣
      ↪cmap=cmap)
```

```
[ ]: <Axes: title={'center': 'Cu interpolated'}>
```



Preprocess geological data Compute distances to structures.

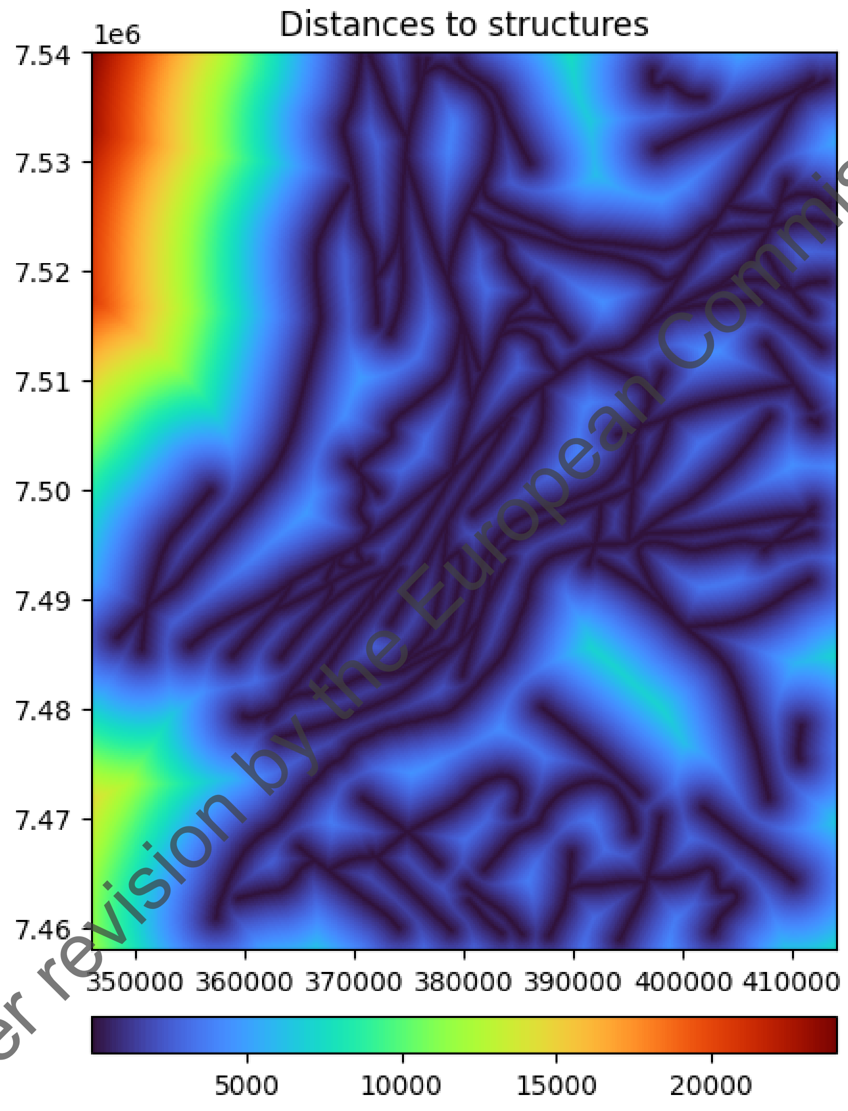
```
[ ]: structures = gpd.read_file(structures_fp)
```

```
[ ]: distances_to_stuctures = distance_computation(geodataframe=structures,␣
       ↪raster_profile=raster_profile)
```

```
[ ]: # Visualize distances to structures
     fig, ax = plt.subplots(figsize=(5, 9))
     cmap = plt.get_cmap('turbo')

     ax.set_title("Distances to structures")
```

```
clrbar = ax.imshow(distances_to_stuctures, cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(distances_to_stuctures, ax = ax, transform = raster_profile["transform"],␣
  ↪cmap=cmap)
```

[ ]:
```
<Axes: title={'center': 'Distances to structures'}>
```



Preprocess geophysical data Unify all geophysical rasters and prepare an anomaly raster from one of them.

[ ]:
```
# First, open all geophysical data rasters and unify them. While the raster␣
  ↪datasets are open,
# we can calculate and collect descriptive statistics for each of them
```

```
with \
    MemoryFile() as memfile, \
    rasterio.open(AEM_inphase_fp) as AEM_inphase, \
    rasterio.open(AEM_quad_fp) as AEM_quad, \
    rasterio.open(AEM_ratio_fp) as AEM_ratio, \
    rasterio.open(Magn_AS_fp) as Magn_AS:
        unified_rasters = unify_raster_grids(AEM_inphase, [AEM_quad, AEM_ratio,␣
 ↪Magn_AS], same_extent=True)
        stats = [
                descriptive_statistics_raster(AEM_inphase),
                descriptive_statistics_raster(AEM_quad),
                descriptive_statistics_raster(AEM_ratio),
                descriptive_statistics_raster(Magn_AS)
        ]

AEM_inphase_data, AEM_inphase_meta = unified_rasters[0]
AEM_quad_data, AEM_quad_meta = unified_rasters[1]
AEM_ratio_data, AEM_ratio_meta = unified_rasters[2]
Magn_AS_data, Magn_AS_meta = unified_rasters[3]
```

```
[ ]: # The AEM_inphase raster has invalid nodata set in its metadata. This could be␣
      ↪noticed later when processing,
     # but in this case it is known beforehand.
     # To fix it, we can assign a new nodata value to the metadata
     AEM_inphase_meta = set_raster_nodata(AEM_inphase_meta, np.min(AEM_inphase_data))
```

```
[ ]: # Statistics for Magnetic raster, which we intend to compute distance to␣
      ↪anomalies for
     stats[3]
```

```
[ ]: {'min': 0.003132963,
      'max': 164.67801,
      'mean': 2.1845575244451223,
      '25%': 0.57473993,
      '50%': 1.1546912,
      '75%': 2.5222025,
      'standard_deviation': 3.0649950135946704,
      'relative_standard_deviation': 1.4030278348349647,
      'skew': 5.535799740817698}
```

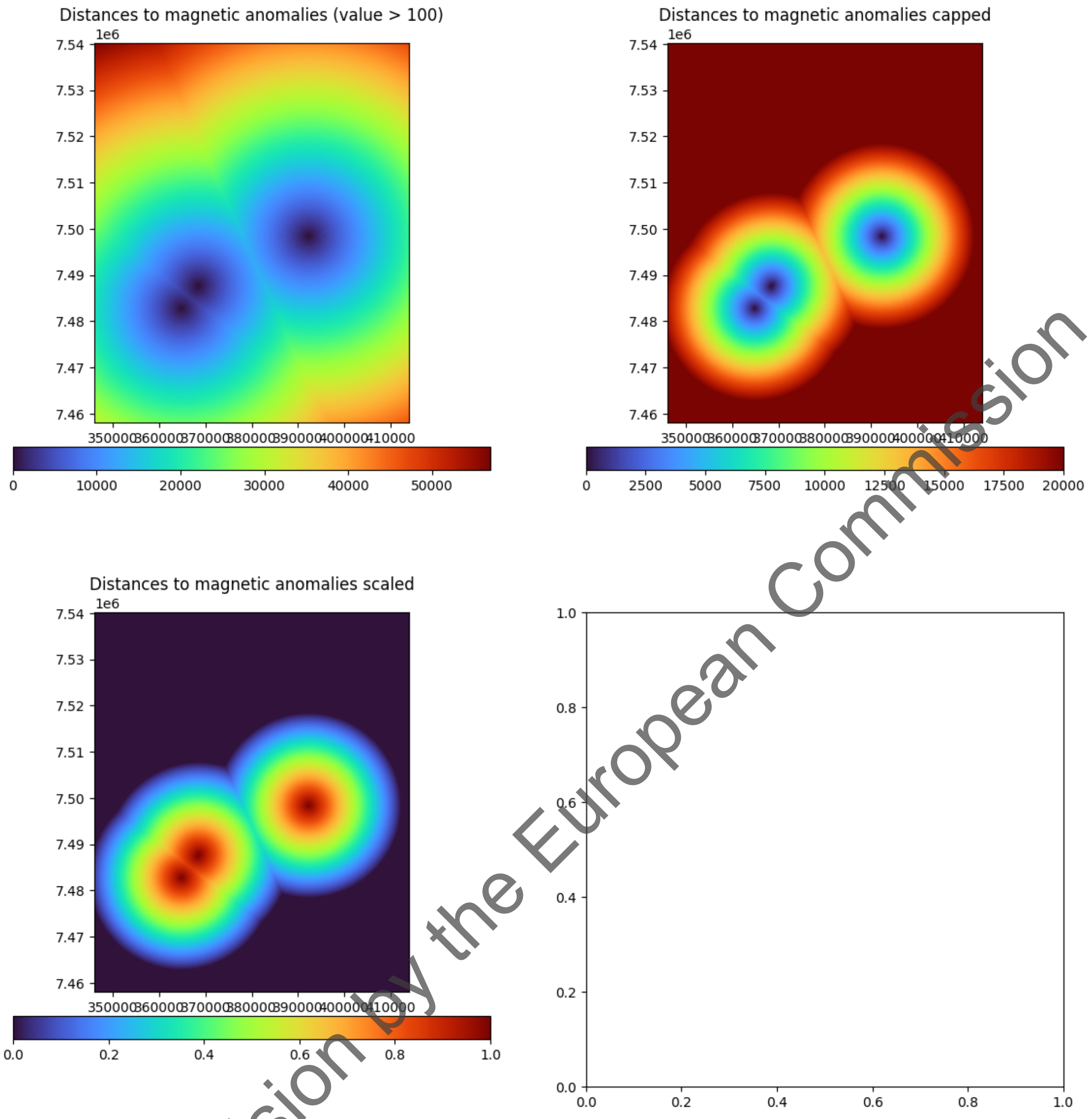
```
[ ]: # Produce anomaly layer from one the rasters

     # Here, 100 is crudely approximated as a potentially interesting threhsold␣
      ↪value for anomalies based on the statistics above:
     # Mean of the data is only as little as 2.18, but max is 164, indicating there␣
      ↪are is a couple of very high pixel cells,
     # i.e. anomalies
```

```
magnetic_anomaly_raster, _ = distance_to_anomaly(raster_profile,␣
  ↪Magn_AS_data[0], 100, "higher")
```

```
[ ]: # To futher process the anomaly layer in preparation for modeling, a maximum␣
       ↪interesting distance can be set and
     # the data values inverted and scaled to range [0, 1]
     magnetic_anomaly_raster_capped = magnetic_anomaly_raster.copy()
     magnetic_anomaly_raster_capped[magnetic_anomaly_raster_capped > 20000] = 20000

     magnetic_anomaly_raster_scaled =␣
       ↪_min_max_scaling(magnetic_anomaly_raster_capped, (1, 0))
```

```
[ ]: # Visualize anomaly raster at different stages
     fig, axs = plt.subplots(2, 2, figsize = (14, 14))
     cmap = plt.get_cmap('turbo')

     axs[0, 0].set_title("Distances to magnetic anomalies (value > 100)")
     clrbar = axs[0, 0].imshow(magnetic_anomaly_raster, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(magnetic_anomaly_raster, ax = axs[0, 0], transform =␣
       ↪raster_profile["transform"], cmap=cmap)

     axs[0, 1].set_title("Distances to magnetic anomalies capped")
     clrbar = axs[0, 1].imshow(magnetic_anomaly_raster_capped, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(magnetic_anomaly_raster_capped, ax = axs[0, 1], transform =␣
       ↪raster_profile["transform"], cmap=cmap)

     axs[1, 0].set_title("Distances to magnetic anomalies scaled")
     clrbar = axs[1, 0].imshow(magnetic_anomaly_raster_scaled, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(magnetic_anomaly_raster_scaled, ax = axs[1, 0], transform =␣
       ↪raster_profile["transform"], cmap=cmap)
```

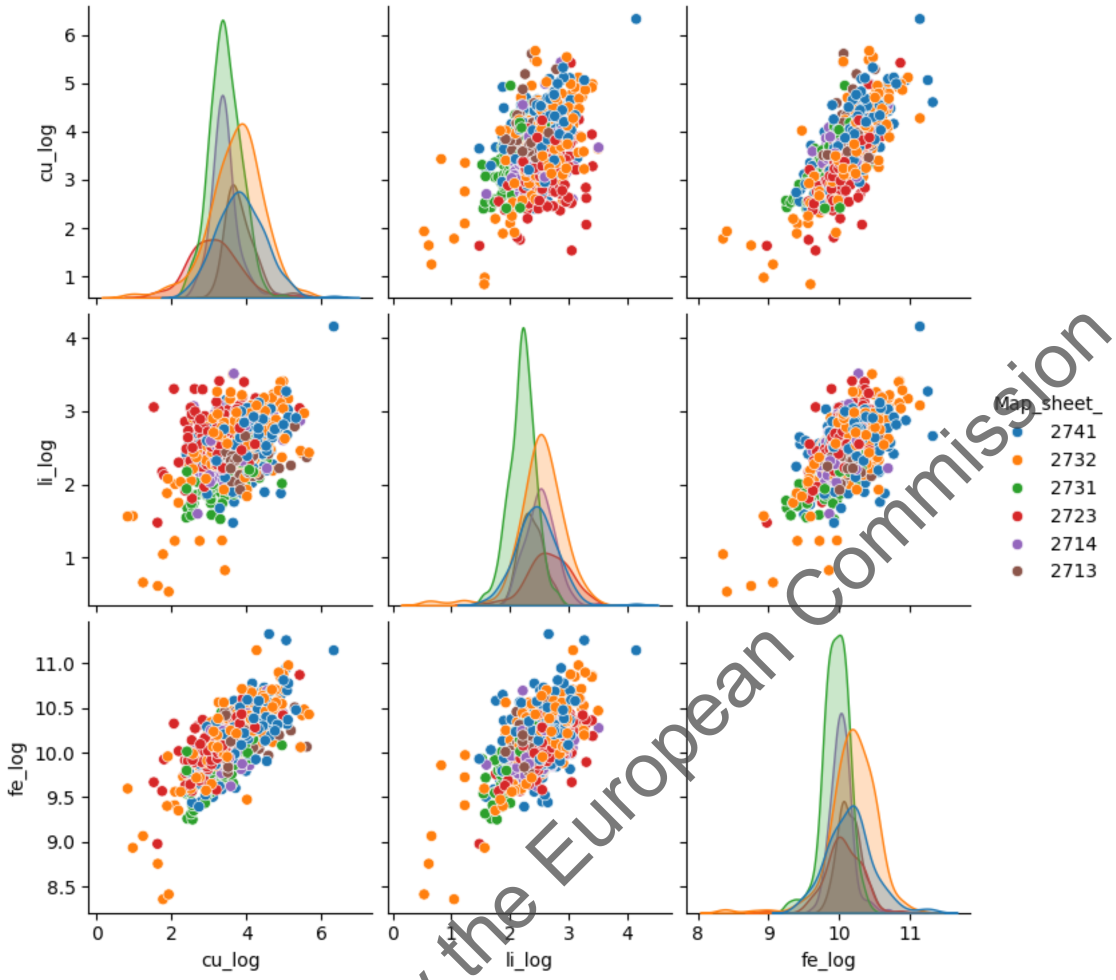
```
[ ]: <Axes: title={'center': 'Distances to magnetic anomalies scaled'}>
```



0.2 3. Explore data

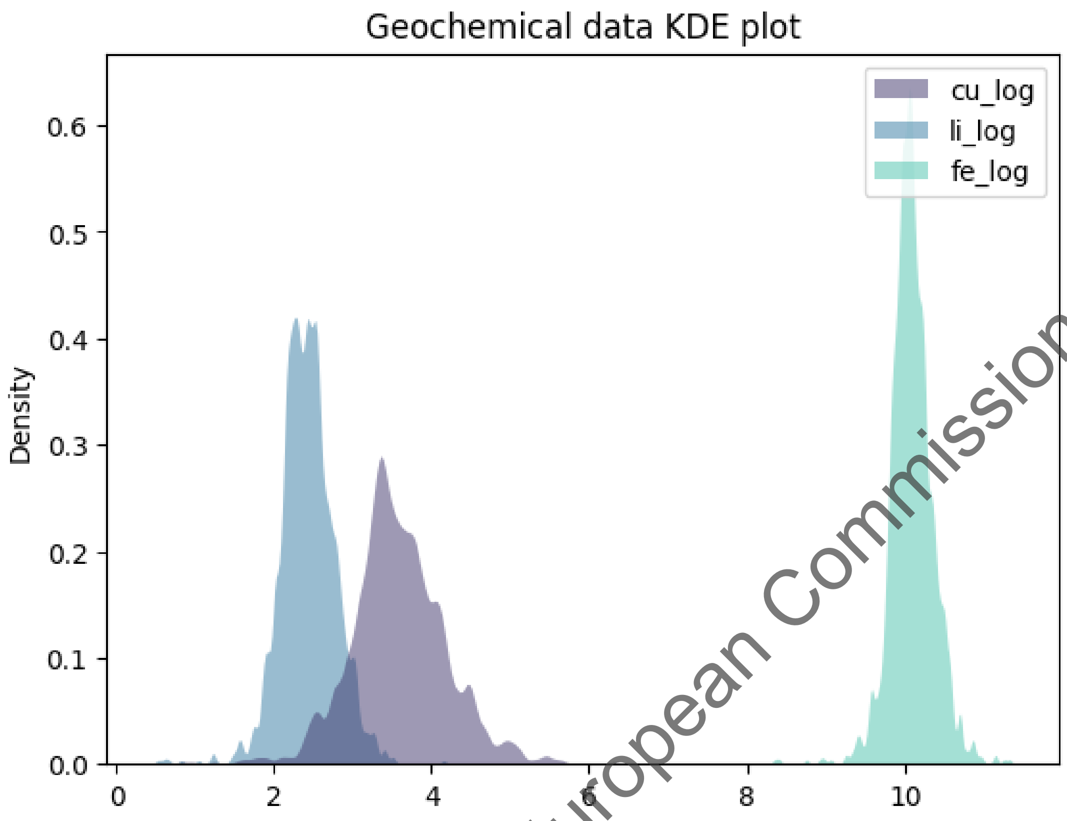
Explore geochemical data Some plot types to visualize vector data are shown.

```
[ ]: # Subset data for plotting
     till_geochem_for_pairplot = till_geochem[["cu_log", "li_log", "fe_log",␣
      ↪"Map_sheet_"]]

     # Plot pairplot of selected log transformed concentrations
     pairplot_grid = pairplot(till_geochem_for_pairplot, hue="Map_sheet_")
```

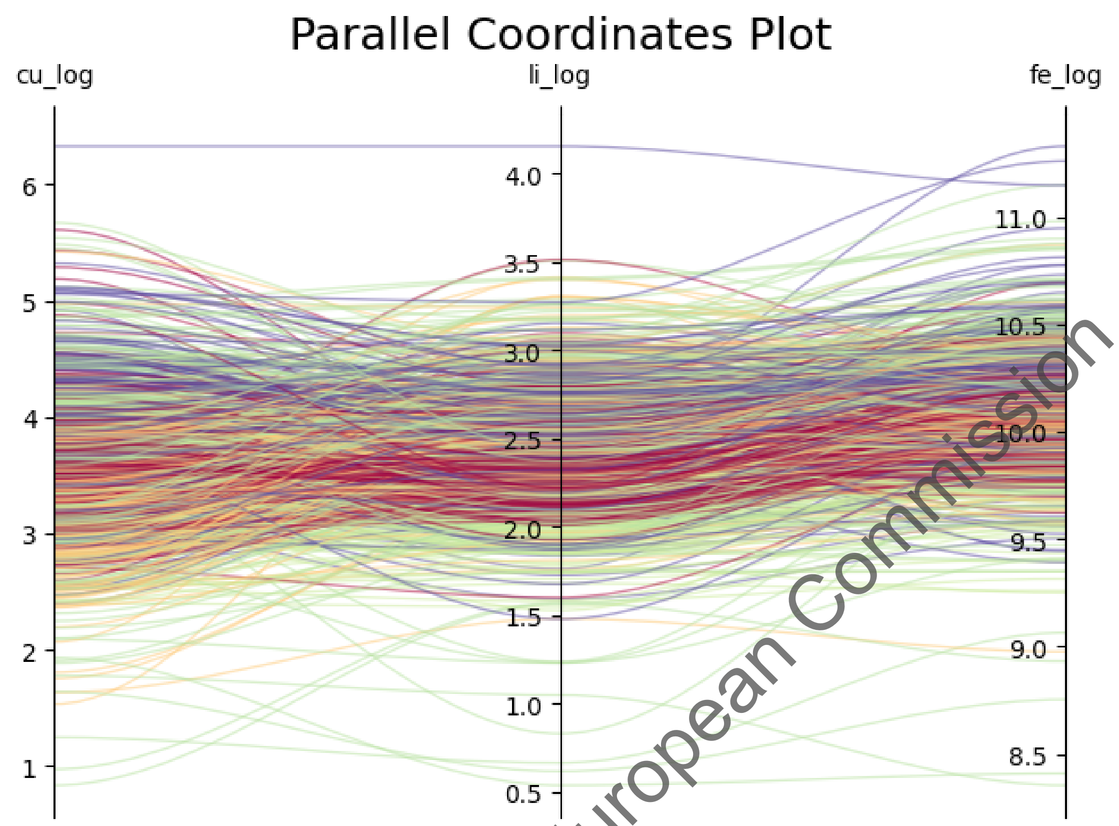


```
[ ]: # Subset data for plotting
     till_geochem_for_kdeplot = till_geochem[["cu_log", "li_log", "fe_log"]]

     # Plot KDE plot of the selected log transformed concentrations
     kdeplot(till_geochem_for_kdeplot, bw_adjust=0.4, fill=True, alpha=0.5,␣
      ↪linewidth=0, palette="mako").set_title("Geochemical data KDE plot")
```

```
[ ]: Text(0.5, 1.0, 'Geochemical data KDE plot')
```



```
[ ]: # Plot parallel coordinates for demonstration purposes. In this case, the plot␣
       ↪does not reveal any particularly
     # interesting patterns from the data
     parallel_coordinates_plot =␣
       ↪plot_parallel_coordinates(till_geochem_for_pairplot,␣
       ↪color_column_name="Map_sheet_")
```



PCA

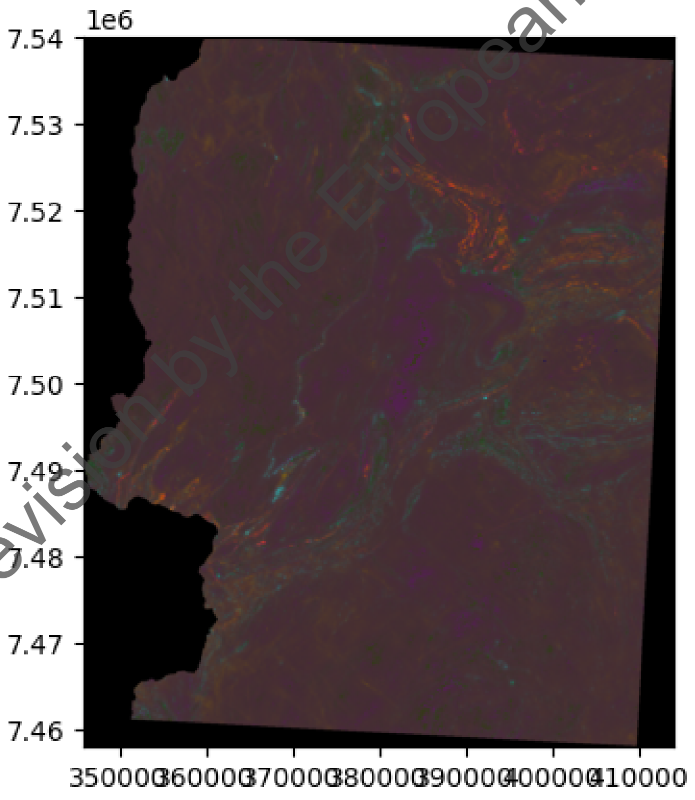
```
[ ]: rasters = [
         (AEM_inphase_data, AEM_inphase_meta),
         (AEM_quad_data, AEM_quad_meta),
         (AEM_ratio_data, AEM_ratio_meta),
         (Magn_AS_data, Magn_AS_meta),
         (distances_to_stuctures, raster_profile)
     ]
     arrays_to_stack = []

     # Process nodata and stack rasters to feed into PCA tool
     for raster_array, raster_profile in unified_rasters:
         arr = raster_array[0] if raster_array.ndim == 3 else raster_array
         arr[arr == raster_profile["nodata"]] = np.nan
         arrays_to_stack.append(arr)

     stacked_arrays = np.stack(arrays_to_stack)
```

```
[ ]: # Compute PCA for the input rasters
     out_array, explained_variances = compute_pca(stacked_arrays, 3)
```

```
explained_variances
```

```
[ ]: array([0.42220193, 0.30625044, 0.17527185])
```

```
[ ]: # Visualize PCA outputs

     # Scale each band
     scaled_bands = []
     for band in range(out_array.shape[0]):
         scaled_band = _min_max_scaling(out_array[band], (0, 255))
         scaled_bands.append(scaled_band)

     # Stack scaled bands back together
     scaled_pca_output = np.stack(scaled_bands)

     # Display the RGB image
     show(scaled_pca_output.astype(np.uint8), transform=raster_profile["transform"])
```



```
[ ]: <Axes: >
```

0.2.1 4. Fuzzy logic modeling

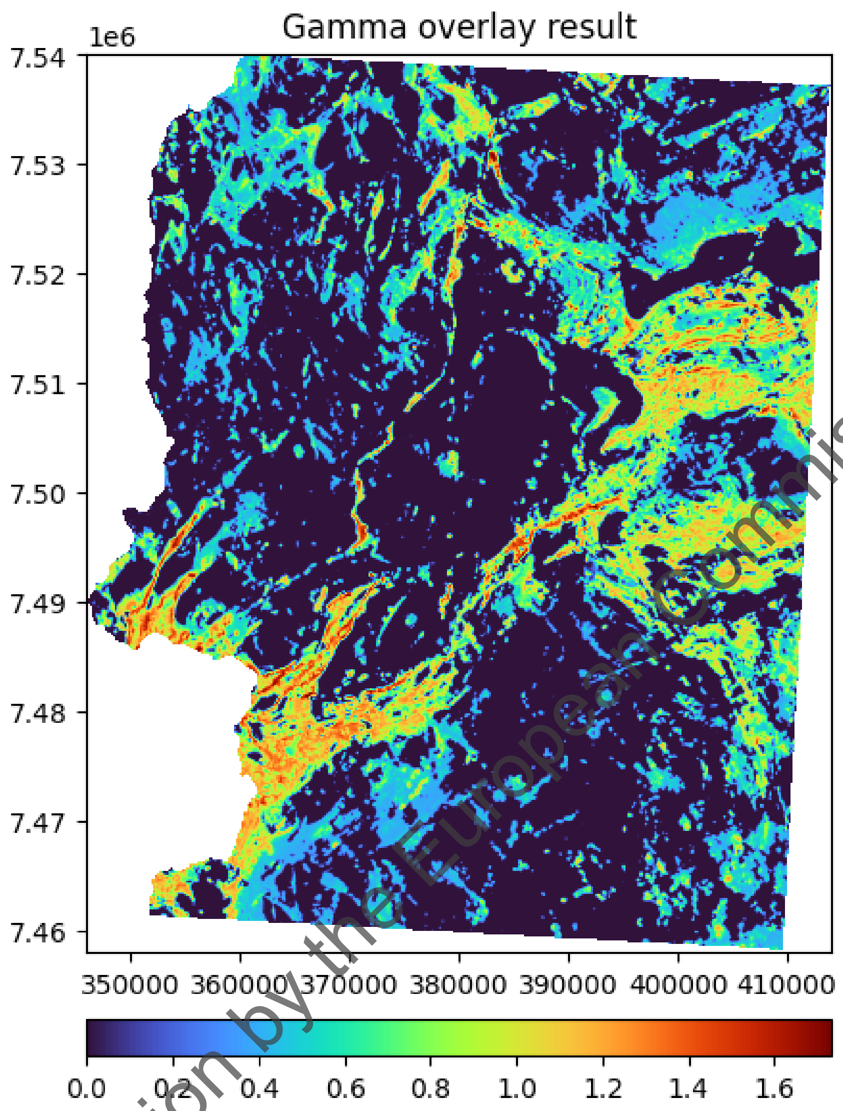
This is a very brief demonstration of fuzzy logic modeling. Usually, fuzzy memberships are produced using various membership functions, but in this all rasters were scaled with the sigmoid transform tool.

```
[ ]: # Transform data before fuzzy overlay
     # We can reuse the 'arrays_to_stack' produced for PCA
     arrays_for_fuzzy = np.stack([_sigmoid_transform(array, (0, 1), 1, True) for␣
      ↪array in arrays_to_stack])
```

```
[ ]: # Compute gamma overlay
     overlay_result = gamma_overlay(arrays_for_fuzzy, 0.5)
```

```
[ ]: # Plot gamma ovelay result
     fig, ax = plt.subplots(1, 1, figsize = (5, 9))

     ax.set_title("Gamma overlay result")
     clrbar = plt.imshow(overlay_result, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(overlay_result, ax = ax, transform = raster_profile["transform"],␣
      ↪cmap=cmap)
```

```
[ ]: <Axes: title={'center': 'Gamma overlay result'}>
```



0.2.2 5. Weights of evidence modeling

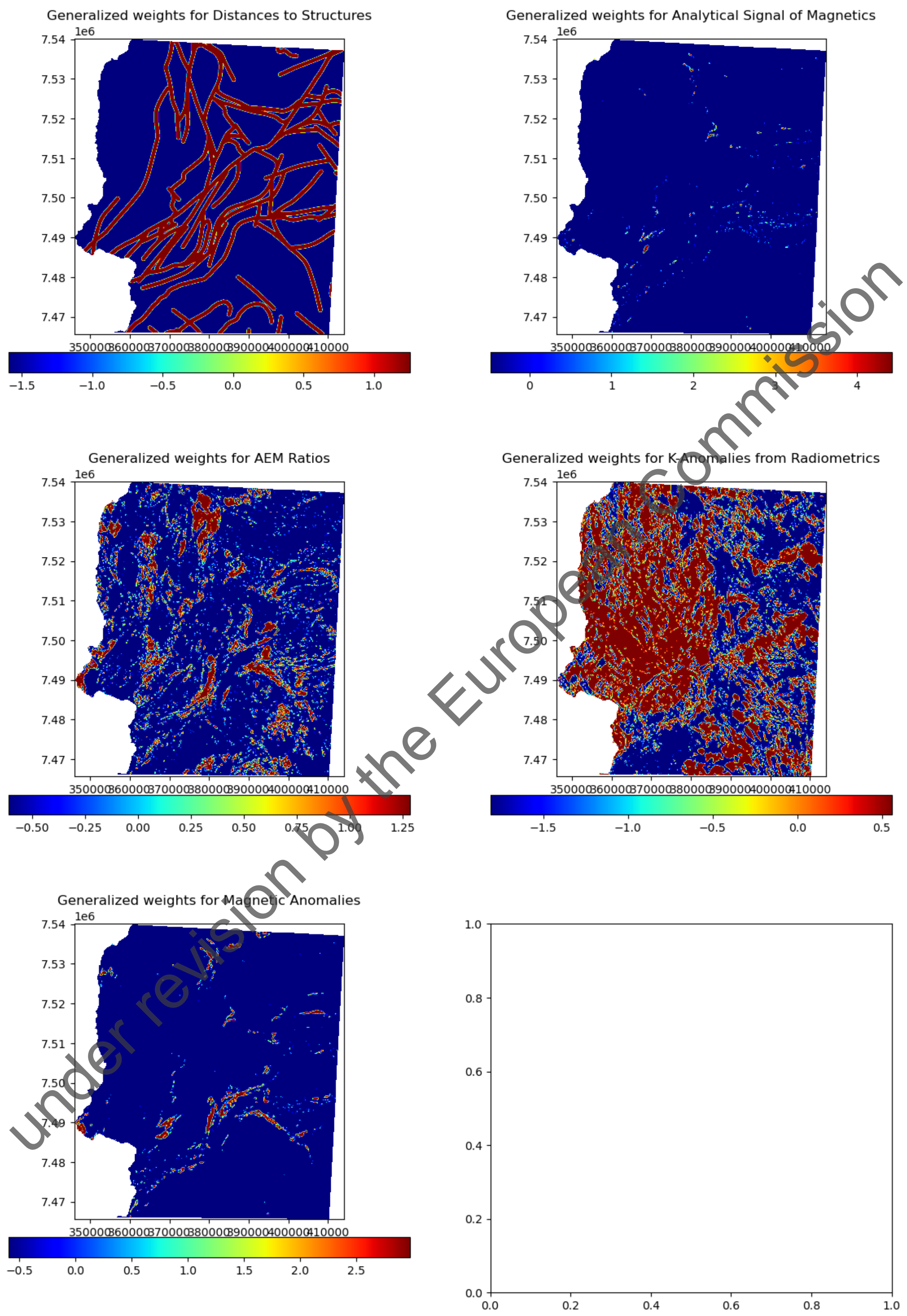
Data for this section can be downloaded from link to data: https://nextcloud.gtk.fi/index.php/s/yqGrRW7sREeoArc.

Please modify the paths accordingly in the corresponding cells of this section

```
[ ]: # Calculate weights

     with rasterio.open("../tests/data/local/workflow_demo/Discretized_data/
      ↪Rcls_Dist_Strucs.tif") as distances_structures, \
          rasterio.open("../tests/data/local/workflow_demo/Discretized_data/
      ↪Rcls_EM_Ratio.tif") as em_ratios, \
          rasterio.open("../tests/data/local/workflow_demo/Discretized_data/
      ↪Rcls_Mag_Anom.tif") as mag_anom, \
          rasterio.open("../tests/data/local/workflow_demo/Discretized_data/
      ↪Rcls_Mag_As.tif") as mag_as, \
          rasterio.open("../tests/data/local/workflow_demo/Discretized_data/
      ↪Rcls_Rd_K.tif") as rad_k:
          deposits = gpd.read_file("../tests/data/local/workflow_demo/
      ↪Discretized_data/IOCG_Deps_Prosp_Occs.shp")

          weights_strucs, arrays_strucs, out_meta, deposit_pixels, evidence_pixels =␣
      ↪weights_of_evidence_calculate_weights(
                  evidential_raster=distances_structures,
                  deposits=deposits,
                  weights_type='ascending',
                  studentized_contrast_threshold=2
          )

          weights_aem_ratio, arrays_aem_ratio, _, _, _ =␣
      ↪weights_of_evidence_calculate_weights(
                  evidential_raster=em_ratios,
                  deposits=deposits,
                  weights_type='ascending',
                  studentized_contrast_threshold=2
          )

          weights_mag_anom, arrays_mag_anom, _, _, _ =␣
      ↪weights_of_evidence_calculate_weights(
                  evidential_raster=mag_anom,
                  deposits=deposits,
                  weights_type='descending',
                  studentized_contrast_threshold=2
          )

          weights_mag_as, arrays_mag_as, _, _, _ =␣
      ↪weights_of_evidence_calculate_weights(
                  evidential_raster=mag_as,
                  deposits=deposits,
                  weights_type='descending',
                  studentized_contrast_threshold=2
          )

          weights_rad_k, arrays_rad_k, _, _, _ =␣
      ↪weights_of_evidence_calculate_weights(
                evidential_raster=rad_k,
                deposits=deposits,
                weights_type='descending',
              studentized_contrast_threshold=2
          )
```

[ ]:
```
# Valize generalized weights arrays
fig, axs = plt.subplots(3, 2, figsize = (14, 20))

axs[0, 0].set_title("Generalized weights for Distances to Structures")
clrbar = axs[0, 0].imshow(arrays_strucs["Generalized W+"], cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(arrays_strucs["Generalized W+"], ax = axs[0, 0], transform =␣
 ↪out_meta["transform"], cmap=cmap)

axs[1, 0].set_title("Generalized weights for AEM Ratios")
clrbar = axs[1, 0].imshow(arrays_aem_ratio["Generalized W+"], cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(arrays_aem_ratio["Generalized W+"], ax = axs[1, 0], transform =␣
 ↪out_meta["transform"], cmap=cmap)

axs[2, 0].set_title("Generalized weights for Magnetic Anomalies")
clrbar = axs[2, 0].imshow(arrays_mag_anom["Generalized W+"], cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(arrays_mag_anom["Generalized W+"], ax = axs[2, 0], transform =␣
 ↪out_meta["transform"], cmap=cmap)

axs[0, 1].set_title("Generalized weights for Analytical Signal of Magnetics")
clrbar = axs[0, 1].imshow(arrays_mag_as["Generalized W+"], cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(arrays_mag_as["Generalized W+"], ax = axs[0, 1], transform =␣
 ↪out_meta["transform"], cmap=cmap)

axs[1, 1].set_title("Generalized weights for K-Anomalies from Radiometrics")
clrbar = axs[1, 1].imshow(arrays_rad_k["Generalized W+"], cmap=cmap)
plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
show(arrays_rad_k["Generalized W+"], ax = axs[1, 1], transform =␣
 ↪out_meta["transform"], cmap=cmap)
```

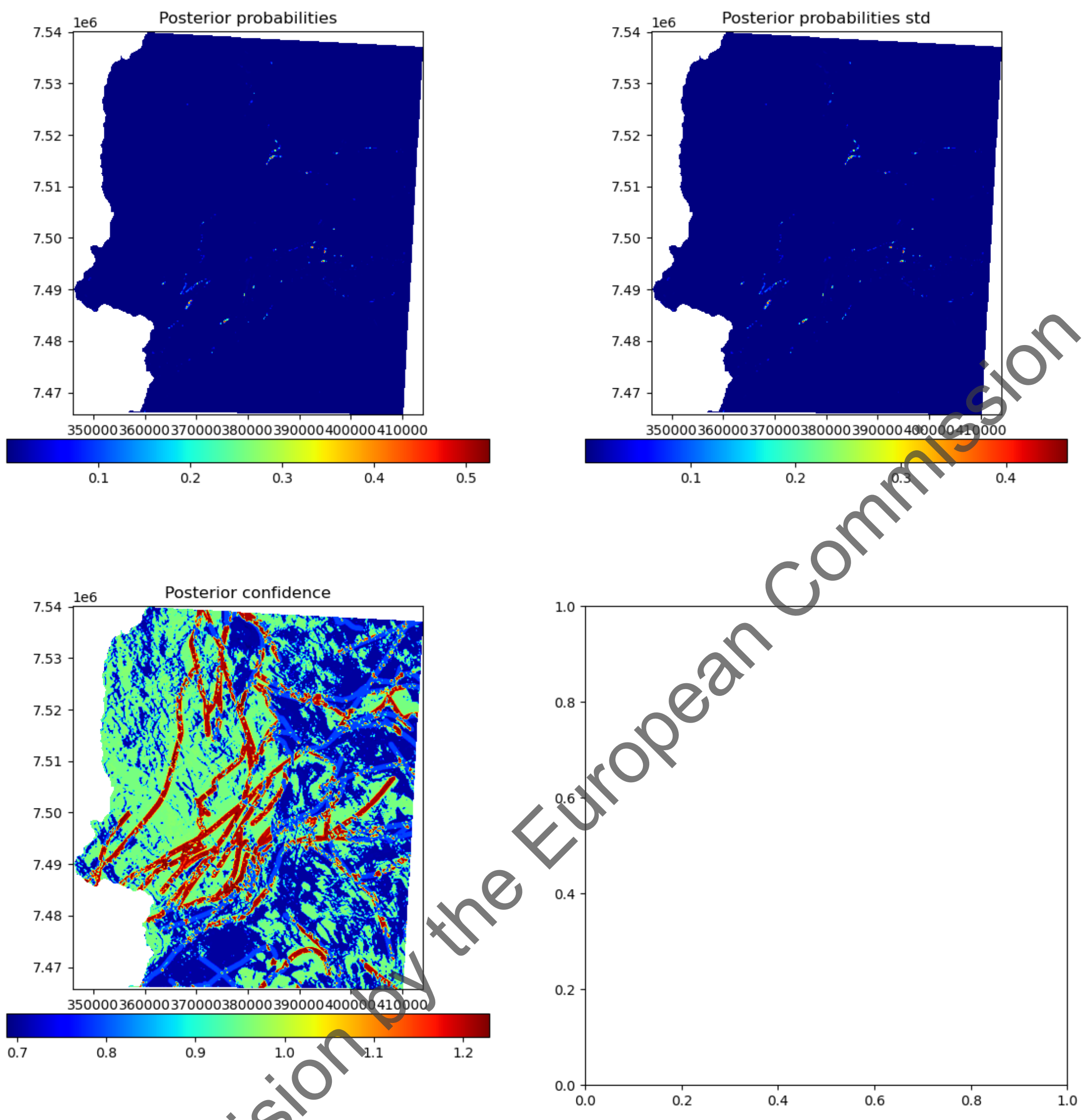
[ ]:
```
'\naxs[1, 0].set_title("Generalized weights for Distances to
Structures")\nclrbar = axs[1, 0].imshow(arrays_desc["W+"],
cmap=cmap)\nplt.colorbar(clrbar, orientation="horizontal", pad =
0.05)\nshow(arrays_desc["W+"], ax = axs[1, 0], transform =
out_meta["transform"], cmap=cmap)\n\naxs[1, 1].set_title("Descending weights -
S_W+")\nclrbar = axs[1, 1].imshow(arrays_desc["S_W+"],
cmap=cmap)\nplt.colorbar(clrbar, orientation="horizontal", pad =
0.05)\nshow(arrays_desc["S_W+"], ax = axs[1, 1], transform =
out_meta["transform"], cmap=cmap)\n\naxs[2, 0].set_title("Descending weights -
Generalized W+")\nclrbar = axs[2, 0].imshow(arrays_desc["Generalized W+"],
cmap=cmap)\nplt.colorbar(clrbar, orientation="horizontal", pad =
0.05)\nshow(arrays_desc["Generalized W+"], ax = axs[2, 0], transform =
out_meta["transform"], cmap=cmap)\n\naxs[2, 1].set_title("Descending weights -
Generalized S_W+")\nclrbar = axs[2, 1].imshow(arrays_desc["Generalized S_W+"],
cmap=cmap)\nplt.colorbar(clrbar, orientation="horizontal", pad =
0.05)\nshow(arrays_desc["Generalized S_W+"], ax = axs[2, 1], transform =
out_meta["transform"], cmap=cmap)\n'
```


Generalized weights for Distances to Structures
Generalized weights for Analytical Signal of Magnetics
Generalized weights for AEM Ratios
Generalized weights for K-Anomalies from Radiometrics
Generalized weights for Magnetic Anomalies
1e6
7.54
7.53
7.52
7.51
7.50
7.49
7.48
7.47
350000
360000
370000
380000
390000
400000
410000

```
[ ]: # Calculate posterior probabilities / responses
     posterior_array, posterior_array_std, posterior_confidence =␣
      ↪weights_of_evidence_calculate_responses([arrays_strucs, arrays_aem_ratio,␣
      ↪arrays_mag_anom, arrays_mag_as, arrays_rad_k], deposit_pixels,␣
      ↪evidence_pixels)
```

```
[ ]: # Plot posterior probabilities weights
     cmap = plt.get_cmap('jet')
     fig, axs = plt.subplots(2, 2, figsize = (14, 14))

     axs[0, 0].set_title("Posterior probabilities")
     clrbar = axs[0, 0].imshow(posterior_array, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(posterior_array, ax = axs[0, 0], transform = out_meta["transform"],␣
      ↪cmap=cmap)

     axs[0, 1].set_title("Posterior probabilities std")
     clrbar = axs[0, 1].imshow(posterior_array_std, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(posterior_array_std, ax = axs[0, 1], transform = out_meta["transform"],␣
      ↪cmap=cmap)

     axs[1, 0].set_title("Posterior confidence")
     clrbar = axs[1, 0].imshow(posterior_confidence, cmap=cmap)
     plt.colorbar(clrbar, orientation="horizontal", pad = 0.05)
     show(posterior_confidence, ax = axs[1, 0], transform = out_meta["transform"],␣
      ↪cmap=cmap)
```

```
[ ]: <Axes: title={'center': 'Posterior confidence'}>
```

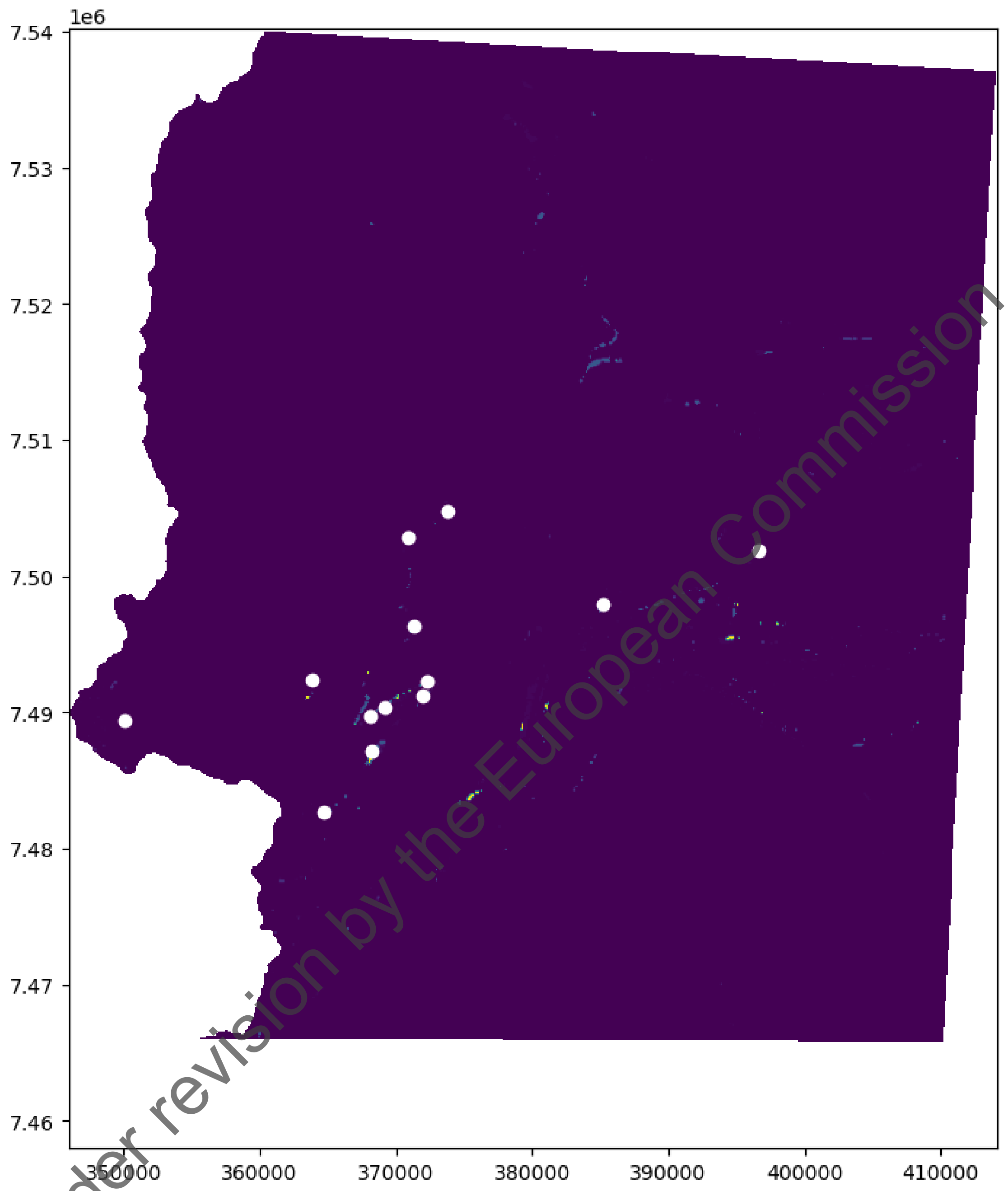


0.2.3 6. Weights of Evidence Results Evaluation

```
[ ]: prospectivity_raster = rasterio.open("../tests/data/local/workflow_demo/
       ↪Discretized_data/Rescale_W_pprb1.tif")
     deposits = gpd.read_file("../tests/data/local/workflow_demo/Discretized_data/
       ↪IOCG_Deps_Prosp_Occs.shp")
```

```
[ ]: fig, ax = plt.subplots(figsize=(10, 10))
     rasterio.plot.show(prospectivity_raster, ax=ax)
     deposits.plot(ax=ax, facecolor='w', edgecolor='w')
```

```
[ ]: <Axes: >
```

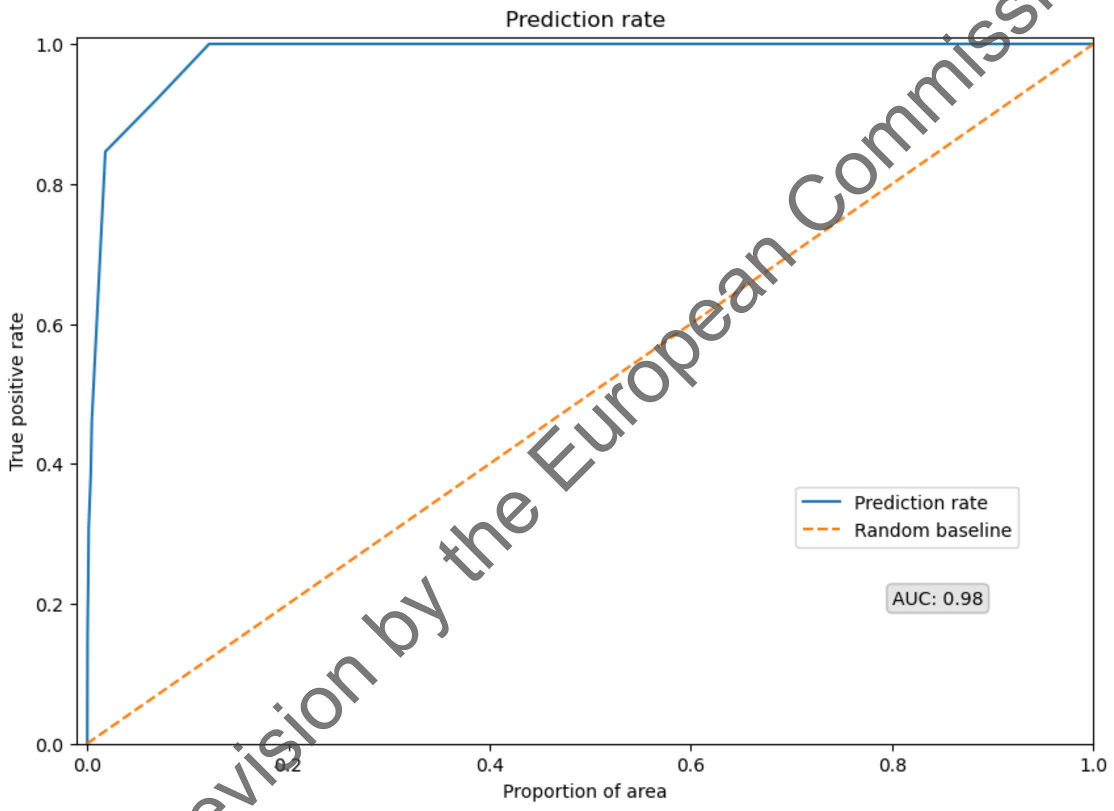


```
[ ]: metrics = calculate_base_metrics(raster=prospectivity_raster, deposits=deposits)
     metrics
```

```
[ ]:    true_positive_rate_values  proportion_of_area_values  threshold_values
     0                   0.000000                   0.000163      1.000000e+00
     1                   0.153846                   0.000443      3.024189e-01
     2                   0.230769                   0.001125      2.756724e-01
```

```
3                    0.307692                  0.001590     4.810836e-02
4                    0.384615                  0.003666     2.851541e-03
5                    0.461538                  0.004707     2.725697e-03
6                    0.846154                  0.018130     5.111912e-04
7                    0.923077                  0.070688     7.635408e-05
8                    1.000000                  0.121263     2.893874e-05
9                    1.000000                  1.000000    -3.402823e+38
```

```
[ ]: p = plot_rate_curve(metrics["proportion_of_area_values"],␣
      ↪metrics["true_positive_rate_values"], "prediction_rate")
```



Appendix 2: EIS Toolkit – Technical Specifications

Funded by
the European Union

EIS Toolkit

None

None

Made by <a href="info@gispo.fi">Gispo Ltd.</a>

Table of contents

1. General

This is the documentation site of the eis_toolkit python package. Here you can find documentation for each module. The documentation is automatically generated from docstrings.

Development of eis_toolkit is related to EIS Horizon EU project.

2. Dependency licenses

Name	Version	License
protobuf	4.24.4	3-Clause BSD License
absl-py	2.0.0	Apache Software License
flatbuffers	23.5.26	Apache Software License
google-auth	2.23.3	Apache Software License
google-auth-oauthlib	1.0.0	Apache Software License
google-pasta	0.2.0	Apache Software License
grpcio	1.59.0	Apache Software License
keras	2.14.0	Apache Software License
libclang	16.0.6	Apache Software License
ml-dtypes	0.2.0	Apache Software License
requests	2.31.0	Apache Software License
rsa	4.9	Apache Software License
tensorboard	2.14.1	Apache Software License
tensorboard-data-server	0.7.2	Apache Software License
tensorflow	2.14.0	Apache Software License
tensorflow-estimator	2.14.0	Apache Software License
tensorflow-io-gcs-filesystem	0.34.0	Apache Software License
tzdata	2023.3	Apache Software License
packaging	23.2	Apache Software License; BSD License
python-dateutil	2.8.2	Apache Software License; BSD License
cligj	0.7.2	BSD
geopandas	0.11.1	BSD
Markdown	3.5	BSD License
MarkupSafe	2.1.3	BSD License
PyKrige	1.7.1	BSD License
Pygments	2.16.1	BSD License
Shapely	1.8.5.post1	BSD License
Werkzeug	3.0.0	BSD License
affine	2.4.0	BSD License
astunparse	1.6.3	BSD License
click	8.1.7	BSD License
click-plugins	1.1.1	BSD License
colorama	0.4.6	BSD License
contourpy	1.1.1	BSD License
cycler	0.12.1	BSD License

Name	Version	License
fiona	1.9.5	BSD License
gast	0.5.4	BSD License
h5py	3.10.0	BSD License
idna	3.4	BSD License
joblib	1.3.2	BSD License
kiwisolver	1.4.5	BSD License
numpy	1.26.1	BSD License
oauthlib	3.2.2	BSD License
pandas	2.1.1	BSD License
patsy	0.5.3	BSD License
pyasn1	0.5.0	BSD License
pyasn1-modules	0.3.0	BSD License
rasterio	1.3.9	BSD License
requests-oauthlib	1.3.1	BSD License
scikit-learn	1.3.2	BSD License
scipy	1.11.3	BSD License
seaborn	0.13.0	BSD License
statsmodels	0.14.0	BSD License
threadpoolctl	3.2.0	BSD License
wrapt	1.14.1	BSD License
eis-toolkit	0.1.0	European Union Public Licence 1.2 (EUPL 1.2)
Pillow	10.1.0	Historical Permission Notice and Disclaimer (HPND)
shellingham	1.5.4	ISC License (ISCL)
imbalanced-learn	0.11.0	MIT
opt-einsum	3.3.0	MIT
snuggs	1.4.7	MIT
GDAL	3.4.3	MIT License
Rtree	1.1.0	MIT License
attrs	23.1.0	MIT License
beartype	0.13.1	MIT License
cachetools	5.3.1	MIT License
charset-normalizer	3.3.1	MIT License
fonttools	4.43.1	MIT License
markdown-it-py	3.0.0	MIT License
mdurl	0.1.2	MIT License

Name	**Version**	**License**
pyparsing	3.1.1	MIT License
pyproj	3.6.1	MIT License
pytz	2023.3.post1	MIT License
rich	13.6.0	MIT License
setuptools-scm	8.0.4	MIT License
six	1.16.0	MIT License
termcolor	2.3.0	MIT License
tomli	2.0.1	MIT License
typer	0.9.0	MIT License
urllib3	2.0.7	MIT License
certifi	2023.7.22	Mozilla Public License 2.0 (MPL 2.0)
matplotlib	3.8.0	Python Software Foundation License
typing_extensions	4.8.0	Python Software Foundation License

3. Conversions

3.1 Convert csv to geodataframe

`csv_to_geodataframe(csv, indexes, target_crs)`

Read CSV file to a GeoDataFrame.

Usage of single index expects valid WKT geometry. Usage of two indexes expects POINT feature(s) X-coordinate as the first index and Y-coordinate as the second index.

Parameters:

Name	Type	Description	Default
`csv`	`Path`	Path to the .csv file to be read.	required
`indexes`	`Sequence[int]`	Index(es) of the geometry column(s).	required
`target_crs`	`int`	Target CRS as an EPSG code.	required

Returns:

Type	Description
`GeoDataFrame`	CSV file read to a GeoDataFrame.

Raises:

Type	Description
`InvalidColumnIndexException`	There is a mismatch between the provided indexes and the shape of the dataframe read from the csv.
`InvalidParameterValueException`	Unable to create a GeoDataFrame with point features from the given input parameters.
`InvalidWktFormatException`	Unable to create a GeoDataFrame of WKT geometry from the given input parameters.

3.2 Convert raster to dataframe

`raster_to_dataframe(raster, bands=None, add_coordinates=False, nodata_value=None)`

Convert raster to Pandas DataFrame.

If bands are not given, all bands are used for conversion. Selected bands are named based on their index e.g., band_1, band_2,...,band_n. If wanted, image coordinates (x, y) for each pixel can be written to dataframe by setting add_coordinates to True.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be converted.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`
`add_coordinates`	`bool`	Determines if pixel coordinates are written into dataframe. Defaults to False.	`False`
`nodata_value`	`Optional[float]`	Specifies the value to be considered as NoData. If None, raster's nodata value is used.	`None`

Returns:

Type	Description
`DataFrame`	Raster converted to a DataFrame.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.

4. Evaluation

4.1 Calculate base metrics

```
calculate_base_metrics(raster, deposits, band=1, negatives=None)
```

Calculate true positive rate, proportion of area and false positive rate values for different thresholds.

Function calculates true positive rate, proportion of area and false positive rate values for different thresholds which are determined from inputted deposit locations and mineral prospectivity map. Note that calculation of false positive rate is optional and is only done if negative point locations are provided.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Mineral prospectivity map or evidence layer.	required
`deposits`	`GeoDataFrame`	Mineral deposit locations as points.	required
`band`	`int`	Band index of the mineral prospectivity map. Defaults to 1.	`1`
`negatives`	`Optional[GeoDataFrame]`	Negative locations as points.	`None`

Returns:

Type	Description
`DataFrame`	DataFrame containing true positive rate, proportion of area, threshold values and false positive rate (optional) values.

Raises:

Type	Description
`NonMatchingCrsException`	The raster and point data are not in the same CRS.
`GeometryTypeException`	The input geometries contain non-point features.

4.2 Classification label evaluation

`summarize_label_metrics_binary(y_true, y_pred)`

Generate a comprehensive report of various evaluation metrics for binary classification results.

The output includes accuracy, precision, recall, F1 scores and confusion matrix elements (true negatives, false positives, false negatives, true positives).

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_pred`	`ndarray`	Predicted labels. The array should come from a binary classifier.	required

Returns:

Type	Description
`Dict[str, Number]`	A dictionary containing the evaluated metrics.

4.3 Classification probability evaluation

`plot_calibration_curve(y_true, y_prob, n_bins=5, plot_title='Calibration curve', ax=None, **kwargs)`

Plot calibration curve (aka realibity diagram).

Calibration curve has the frequency of the positive labels on the y-axis and the predicted probability on the x-axis. Generally, the close the calibration curve is to line x=y, the better the model is calibrated.

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required
`plot_title`	`Optional[str]`	Title for the plot. Defaults to "Precision-Recall curve".	`'Calibration curve'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to matplotlib.pyplot.plot.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes containing the plot.

`plot_det_curve(y_true, y_prob, plot_title='DET curve', ax=None, **kwargs)`

Plot DET (detection error tradeoff) curve.

DET curve is a binary classification multi-threshold metric. DET curves are a variation of ROC curves where False Negative Rate is plotted on the y-axis instead of True Positive Rate. The ideal performance corner of the plot is bottom-left. When comparing the performance of different models, DET curves can be slightly easier to assess visually than ROC curves.

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required
`plot_title`	`Optional[str]`	Title for the plot. Defaults to "DET curve".	`'DET curve'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to matplotlib.pyplot.plot.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes containing the plot.

`plot_precision_recall_curve(y_true, y_prob, plot_title='Precision-Recall curve', ax=None, **kwargs)`

Plot precision-recall curve.

Precision-recall curve is a binary classification multi-threshold metric. Precision-recall curve shows the tradeoff between precision and recall for different classification thresholds. It can be a useful measure of success when classes are imbalanced.

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required
`plot_title`	`Optional[str]`	Title for the plot. Defaults to "Precision-Recall curve".	`'Precision-Recall curve'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to matplotlib.pyplot.plot.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes containing the plot.

`plot_predicted_probability_distribution(y_prob, n_bins=5, plot_title='Distribution of predicted probabilities', ax=None, **kwargs)`

Plot a histogram of the predicted probabilities.

Parameters:

Name	Type	Description	Default
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required
`n_bins`	`int`	Number of bins used for the histogram. Defaults to 5.	`5`
`plot_title`	`Optional[str]`	Title for the plot. Defaults to "Distribution of predicted probabilities".	`'Distribution of predicted probabilities'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to sns.histplot and matplotlib.	`{}`

Returns:

Type	Description
`Axes`	Matplolib axes containing the plot.

`plot_roc_curve(y_true, y_prob, plot_title='ROC curve', ax=None, **kwargs)`

Plot ROC (receiver operating characteristic) curve.

ROC curve is a binary classification multi-threshold metric. The ideal performance corner of the plot is top-left. AUC of the ROC curve summarizes model performance across different classification thresholds.

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required
`plot_title`	`Optional[str]`	Title for the plot. Defaults to "ROC curve".	`'ROC curve'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to matplotlib.pyplot.plot.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes containing the plot.

`summarize_probability_metrics(y_true, y_prob)`

Generate a comprehensive report of various evaluation metrics for classification probabilities.

The output includes ROC AUC, log loss, average precision and Brier score loss.

Parameters:

Name	Type	Description	Default
`y_true`	`ndarray`	True labels.	required
`y_prob`	`ndarray`	Predicted probabilities for the positive class. The array should come from a binary classifier.	required

Returns:

Type	Description
`Dict[str, float]`	A dictionary containing the evaluated metrics.

4.4 Plot confusion matrix

```
plot_confusion_matrix(confusion_matrix, cmap=None, plot_title='Confusion matrix', ax=None, **kwargs)
```

Plot confusion matrix to visualize classification results.

Parameters:

Name	Type	Description	Default
`confusion_matrix`	`ndarray`	The confusion matrix as 2D Numpy array. Expects the first element (upper-left corner) to have True negatives.	required
`cmap`	`Optional[Union[str, Colormap, Sequence]]`	Colormap name, matplotlib colormap objects or list of colors for coloring the plot. Optional parameter.	`None`
`plot_title`	`str`	Title for the plot. Defaults to "Confusion matrix".	`'Confusion matrix'`
`ax`	`Optional[Axes]`	An existing Axes in which to draw the plot. Defaults to None.	`None`
`**kwargs`		Additional keyword arguments passed to sns.heatmap.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes containing the plot.

Raises:

Type	Description
`InvalidDataShapeException`	Raised if input confusion matrix is not square.

4.5 Plot neural network training performance (accuracy and loss)

`plot_nn_model_accuracy(model_history)`

Plot training and validation accuracies for a neural network model.

Parameters:

Name	Type	Description	Default
`model_history`	`dict`	Dictionary containing neural network model training history information, specifically entries for "accuracy" and "val_accuracy".	required

Returns:

Type	Description
`Axes`	Matplotlib axes containing the produced plot.

Raises:

Type	Description
`InvalidDatasetException`	Raised if "accuracy" or "val_accuracy" are not found in the model_history.
`InvalidDataShapeException`	Raised if "accuracy" and "val_accuracy" have mismatching lengths.

`plot_nn_model_loss(model_history)`

Plot training and validation losses for a neural network model.

Parameters:

Name	Type	Description	Default
`model_history`	`dict`	Dictionary containing neural network model training history information, specifically entries for "loss" and "val_loss".	required

Returns:

Type	Description
`Axes`	Matplotlib axes containing the produced plot.

Raises:

Type	Description
`InvalidDatasetException`	Raised if "loss" or "val_loss" are not found in the model_history.
`InvalidDataShapeException`	Raised if "loss" and "val_loss" have mismatching lengths.

4.6 Plot prediction-area (P-A) curves

`plot_prediction_area_curves(true_positive_rate_values, proportion_of_area_values, threshold_values)`

Plot prediction-area (P-A) plot.

Plots prediction area plot that can be used to evaluate mineral prospectivity maps and evidential layers. See e.g., Yousefi and Carranza (2015).

The inputs needed for this tool can be obtained with calculate_base_metrics() tool.

Parameters:

Name	Type	Description	Default
`true_positive_rate_values`	`Union[ndarray, Series]`	True positive rate values.	required
`proportion_of_area_values`	`Union[ndarray, Series]`	Proportion of area values.	required
`threshold_values`	`Union[ndarray, Series]`	Threshold values.	required

Returns:

Type	Description
`Figure`	P-A plot figure object.

Raises:

Type	Description
`InvalidParameterValueException`	true_positive_rate_values or proportion_of_area_values values are out of bounds.

References

Yousefi, Mahyar, and Emmanuel John M. Carranza. "Fuzzification of continuous-value spatial evidence for mineral prospectivity mapping." Computers & Geosciences 74 (2015): 97-109.

4.7 Plot rate curve

```
plot_rate_curve(x_values, y_values, plot_title='success_rate')
```

Plot success rate.

Y-axis is true positive rate and x-axis is proportion of area.

Parameters:

Name	Type	Description	Default
`x_values`	`Union[ndarray, Series]`	Proportion of area values.	required
`y_values`	`Union[ndarray, Series]`	True positive rate values.	required
`plot_title`	`str`	Success rate	`'success_rate'`

Returns:

Type	Description
`Figure`	Matplotlib figure containing the produced plot.

Raises:

Type	Description
`InvalidParameterValueException`	Invalid plot type.
`InvalidParameterValueException`	x_values or y_values are out of bounds.

5. Exploratory analyses

5.1 Chi-square test

`chi_square_test(data, target_column, columns=None)`

Perform a Chi-square test of independence between a target variable and one or more other variables.

Input data should be categorical data. Continuous data or non-categorical data should be discretized or binned before using this function, as Chi-square tests are not applicable to continuous variables directly.

The test assumes that the observed frequencies in each category are independent.

Parameters:

Name	Type	Description	Default
`data`	`DataFrame`	Dataframe containing the input data.	required
`target_column`	`str`	Variable against which independence of other variables is tested.	required
`columns`	`Optional[Sequence[str]]`	Variables that are tested against the variable in target_column. If None, every column is used.	`None`

Returns:

Type	Description
`Dict[str, Dict[str, float]]`	Test statistics, p-value and degrees of freedom for each variable.

Raises:

Type	Description
`EmptyDataFrameException`	Input Dataframe is empty.
`InvalidParameterValueException`	Invalid column is input.

5.2 Correlation matrix

`correlation_matrix(data, columns=None, correlation_method='pearson', min_periods=None)`

Compute correlation matrix on the input data.

It is assumed that the data is numeric, i.e. integers or floats. NaN values are excluded from the calculations.

Parameters:

Name	Type	Description	Default
`data`	`DataFrame`	Dataframe containing the input data.	required
`columns`	`Optional[Sequence[str]]`	Columns to include in the correlation matrix. If None, all numeric columns are used.	`None`
`correlation_method`	`Literal[pearson, kendall, spearman]`	'pearson', 'kendall', or 'spearman'. Defaults to 'pearson'.	`'pearson'`
`min_periods`	`Optional[int]`	Minimum number of observations required per pair of columns to have valid result. Optional.	`None`

Returns:

Type	Description
`DataFrame`	Dataframe containing matrix representing the correlation coefficient between the corresponding pair of variables.

Raises:

Type	Description
`EmptyDataFrameException`	The input Dataframe is empty.
`InvalidParameterValueException`	min_periods argument is used with method 'kendall'.
`NonNumericDataException`	The selected columns contain non-numeric data.

`plot_correlation_matrix(matrix, annotate=True, cmap=None, plot_title=None, **kwargs)`

Create a Seaborn heatmap to visualize correlation matrix.

Parameters:

Name	Type	Description	Default
`matrix`	`DataFrame`	Correlation matrix as a DataFrame.	required
`annotate`	`bool`	If plot squares should display the correlation values. Defaults to True.	`True`
`cmap`	`Optional[str]`	Colormap name for plotting. Optional parameter. Defaults to None, in which case a default colormap is used.	`None`
`plot_title`	`Optional[str]`	Title of the plot. Optional parameter, defaults to none (no title).	`None`
`**kwargs`	`dict`	Additional parameters to pass to Seaborn and matplotlib.	`{}`

Returns:

Type	Description
`Axes`	Matplotlib axes object with the produced plot.

Raises:

Type	Description
`EmptyDataFrameException`	Input matrix is empty.

5.3 Covariance matrix

`covariance_matrix(data, columns=None, min_periods=None, delta_degrees_of_freedom=1)`

Compute covariance matrix on the input data.

It is assumed that the data is numeric, i.e. integers or floats. NaN values are excluded from the calculations.

Parameters:

Name	Type	Description	Default
`data`	`DataFrame`	Dataframe containing the input data.	required
`columns`	`Optional[Sequence[str]]`	Columns to include in the covariance matrix. If None, all numeric columns are used.	`None`
`min_periods`	`Optional[int]`	Minimum number of observations required per pair of columns to have valid result. Optional.	`None`
`delta_degrees_of_freedom`	`int`	Delta degrees of freedom used for computing covariance matrix. Defaults to 1.	`1`

Returns:

Type	Description
`DataFrame`	Dataframe containing matrix representing the covariance between the corresponding pair of variables.

Raises:

Type	Description
`EmptyDataFrameException`	The input Dataframe is empty.
`InvalidParameterValueException`	Provided value for delta_degrees_of_freedom or min_periods is negative.
`NonNumericDataException`	The input data contain non-numeric data.

5.4 DBSCAN

`dbscan_array(data, max_distance=0.5, min_samples=5)`

Perform DBSCAN clustering on Numpy array data.

If the bands/datasets that form the input 3D Numpy array have different scales and represent different phenomena, consider normalizing or standardizing data before running DBSCAN to avoid biased clusters.

Note that the results depend heavily on the parameter values that might require careful tuning. Note also that clustering can be computationally intesive for large datasets, for highly dimensional data consider dimensionality reduction techiniques such as PCA.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	A 3D Numpy array containing the input data. Expects data to be stacked 2D arrays with shape (bands, height, width).	required
`max_distance`	`Number`	The maximum distance between two samples for one to be considered as in the neighborhood of the other. Defaults to 0.5.	`0.5`
`min_samples`	`int`	The number of samples in a neighborhood for a point to be considered as a core point. Defaults to 5.	`5`

Returns:

Type	Description
`ndarray`	Clustering results as a 2D cluster labels array.

Raises:

Type	Description
`EmptyDataException`	The input Numpy array is empty.
`InvalidDataShapeException`	Input data has incorrect number of dimensions (other than 3).
`InvalidParameterException`	The maximum distance between two samples in a neighborhood is not greater than zero or the number of samples in a neighborhood is not greater than one.

`dbscan_vector(data, include_coordinates=True, columns=None, max_distance=0.5, min_samples=5)`

Perform DBSCAN clustering on a Geodataframe.

The attributes to include in clustering can be controlled with `include_coordinates` and `columns` parameters. Coordinates will add spatial proximity and columns the selected attributes in the cluster creation process. If coordinates are omitted, at least some columns need to be included.

If columns are included and the attributes have different scales and represent different phenomena, consider normalizing or standardizing data before running DBSCAN to avoid biased clusters.

Note that the results depend heavily on the parameter values that might require careful tuning. Note also that clustering can be computationally intesive for large datasets, for highly dimensional data consider dimensionality reduction techniques such as PCA.

Parameters:

Name	Type	Description	Default
`data`	`GeoDataFrame`	GeoDataFrame containing the input data.	required
`include_coordinates`	`bool`	If feature coordinates (spatial proximity) will be included in the clustering process. Defaults to True.	`True`
`columns`	`Optional[Sequence[str]]`	Columns/attributes in the input Geodataframe to be included in the clustering process. Optional parameter, defaults to no columns included (except coordinates).	`None`
`max_distance`	`Number`	The maximum distance between two samples for one to be considered as in the neighborhood of the other. Defaults to 0.5.	`0.5`
`min_samples`	`int`	The number of samples in a neighborhood for a point to be considered as a core point. Defaults to 5.	`5`

Returns:

Type	Description
`GeoDataFrame`	GeoDataFrame containing new column for assigned cluster labels.

Raises:

Type	Description
`EmptyDataFrameException`	The input GeoDataFrame is empty.
`InvalidColumnException`	All specified columns were not found in the input GeoDataFrame.
`InvalidParameterException`	The maximum distance between two samples in a neighborhood is not greater than zero, the number of samples in a neighborhood is not greater than one or or both coordinates and attributes are omitted.

5.5 Descriptive statistics

`descriptive_statistics_dataframe(input_data, column)`

Generate descriptive statistics from vector data.

Generates min, max, mean, quantiles(25%, 50% and 75%), standard deviation, relative standard deviation and skewness.

Parameters:

Name	Type	Description	Default
`input_data`	`Union[DataFrame, GeoDataFrame]`	Data to generate descriptive statistics from.	required
`column`	`str`	Specify the column to generate descriptive statistics from.	required

Returns:

Type	Description
`dict`	The descriptive statistics in previously described order.

`descriptive_statistics_raster(input_data)`

Generate descriptive statistics from raster data.

Generates min, max, mean, quantiles(25%, 50% and 75%), standard deviation, relative standard deviation and skewness. Nodata values are removed from the data before the statistics are computed.

Parameters:

Name	Type	Description	Default
`input_data`	`DatasetReader`	Data to generate descriptive statistics from.	required

Returns:

Type	Description
`dict`	The descriptive statistics in previously described order.

5.6 Feature importance

`evaluate_feature_importance(model, x_test, y_test, feature_names, n_repeats=50, random_state=None)`

Evaluate the feature importance of a sklearn classifier or regressor.

Parameters:

Name	Type	Description	Default
`model`	`BaseEstimator`	A trained and fitted Sklearn model.	required
`x_test`	`ndarray`	Testing feature data (X data need to be normalized / standardized).	required
`y_test`	`ndarray`	Testing label data.	required
`feature_names`	`Sequence[str]`	Names of the feature columns.	required
`n_repeats`	`int`	Number of iteration used when calculate feature importance. Defaults to 50.	`50`
`random_state`	`Optional[int]`	random state for repeatability of results. Optional parameter.	`None`

Returns:

Type	Description
`DataFrame`	A dataframe containing features and their importance.
`dict`	A dictionary containing importance mean, importance std, and overall importance.

Raises:

Type	Description
`InvalidDatasetException`	Either array is empty.
`InvalidParameterValueException`	Value for 'n_repeats' is not at least one.

5.7 K-means clustering

`k_means_clustering_array(data, number_of_clusters=None, random_state=None)`

Perform k-means clustering on Numpy array data.

If the bands/datasets that form the input 3D Numpy array have different scales and represent different phenomena, consider normalizing or standardizing data before running k-means to avoid biased clusters.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	A 3D Numpy array containing the input data. Expects data to be stacked 2D arrays with shape (bands, height, width).	required
`number_of_clusters`	`Optional[int]`	The number of clusters (>= 1) to form. Optional parameter. If not provided, optimal number of clusters is computed using the elbow method.	`None`
`random_state`	`Optional[int]`	A random number generation for centroid initialization to make the randomness deterministic. Optional parameter.	`None`

Returns:

Type	Description
`ndarray`	Clustering results as a 2D cluster labels array.

Raises:

Type	Description
`EmptyDataException`	The input Numpy array is empty.
`InvalidDataShapeException`	Input data has incorrect number of dimensions (other than 3).
`InvalidParameterException`	The number of clusters is less than one.

`k_means_clustering_vector(data, include_coordinates=True, columns=None, number_of_clusters=None, random_state=None)`

Perform k-means clustering on a Geodataframe.

The attributes to include in clustering can be controlled with `include_coordinates` and `columns` parameters. Coordinates will add spatial proximity and columns the selected attributes in the cluster creation process. If coordinates are omitted, at least some columns need to be included.

If columns are included and the attributes have different scales and represent different phenomena, consider normalizing or standardizing data before running k-means to avoid biased clusters.

Parameters:

Name	Type	Description	Default
`data`	`GeoDataFrame`	A GeoDataFrame containing the input data.	required
`include_coordinates`	`bool`	If feature coordinates (spatial proximity) will be included in the clustering process. Defaults to True.	`True`
`columns`	`Optional[Sequence[str]]`	Columns/attributes in the input Geodataframe to be included in the clustering process. Optional parameter, defaults to no columns included (except coordinates).	`None`
`number_of_clusters`	`Optional[int]`	The number of clusters (>= 1) to form. Optional parameter. If not provided, optimal number of clusters is computed using the elbow method.	`None`
`random_state`	`Optional[int]`	A random number generation for centroid initialization to make the randomness deterministic. Optional parameter.	`None`

Returns:

Type	Description
`GeoDataFrame`	GeoDataFrame containing assigned cluster labels.

Raises:

Type	Description
`EmptyDataFrameException`	The input GeoDataFrame is empty.
`InvalidParameterException`	The number of clusters is less than one or both coordinates and attributes are omitted.
`InvalidColumnException`	All specified columns were not found in the input GeoDataFrame.

5.8 Local Moran's I

```
local_morans_i(gdf, column, weight_type='queen', k=4, permutations=999)
```

Execute Local Moran's I calculation for the data.

Parameters:

Name	Type	Description	Default
`gdf`	`GeoDataFrame`	The geodataframe that contains the data to be examined with local morans I.	required
`column`	`str`	The column to be used in the analysis.	required
`weight_type`	`Literal[queen, knn]`	The type of spatial weights matrix to be used. Defaults to "queen".	`'queen'`
`k`	`int`	Number of nearest neighbors for the KNN weights matrix. Defaults to 4.	`4`
`permutations`	`int`	Number of permutations for significance testing. Defaults to 999.	`999`

Returns:

Type	Description
`GeoDataFrame`	Geodataframe appended with two new columns: one with Local Moran's I statistic and one with p-value for the statistic.

Raises:

Type	Description
`EmptyDataFrameException`	The input geodataframe is empty.

5.9 Normality test

`normality_test_array(data, bands=None, nodata_value=None)`

Compute Shapiro-Wilk test for normality on the input Numpy array.

It is assumed that 3D input array represents multiband raster and the first dimension is the number of bands (same shape as Rasterio reads a raster into an array). Normality is calculated for each band separately. NaN values and optionally a specified nodata value are masked out before calculations.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	Numpy array containing the input data. Array should either be 1D, 2D or 3D.	required
`bands`	`Optional[Sequence[int]]`	Band selection. Applies only if input array is 3D. If None, normality is tested for each band.	`None`
`nodata_value`	`Optional[Number]`	Nodata value to be masked out. Optional parameter.	`None`

Returns:

Type	Description
`Dict[str, Dict[str, float]]`	Test statistic and p_value for each selected band in a dictionary.

Raises:

Type	Description
`EmptyDataException`	The input data is empty.
`InvalidRasterBandException`	All selected bands were not found in the input data.
`InvalidDataShapeException`	Input data has incorrect number of dimensions (> 3).
`SampleSizeExceededException`	Input data exceeds the maximum of 5000 samples.

`normality_test_dataframe(data, columns=None)`

Compute Shapiro-Wilk test for normality on the input DataFrame.

Nodata values are dropped automatically.

Parameters:

Name	Type	Description	Default
`data`	`DataFrame`	Dataframe containing the input data.	required
`columns`	`Optional[Sequence[str]]`	Column selection. If none, normality is tested for all columns.	`None`

Returns:

Type	Description
`Dict[str, Dict[str, float]]`	Test statistic and p_value for each selected column in a dictionary.

Raises:

Type	Description
`EmptyDataException`	The input data is empty.
`InvalidColumnException`	All selected columns were not found in the input data.
`NonNumericDataException`	Selected columns contain non-numeric data or no numeric columns were found.
`SampleSizeExceededException`	Input data exceeds the maximum of 5000 samples.

5.10 Plot parallel coordinates

`plot_parallel_coordinates(df, color_column_name, plot_title=None, palette_name=None, curved_lines=True)`

Plot a parallel coordinates plot.

Automatically removes all rows containing null/nan values. Tries to convert columns to numeric to be able to plot them. If more than 8 columns are present (after numeric filtering), keeps only the first 8 to plot.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	The DataFrame to plot.	required
`color_column_name`	`str`	The name of the column in df to use for color encoding.	required
`plot_title`	`Optional[str]`	The title for the plot. Default is None.	`None`
`palette_name`	`Optional[str]`	The name of the color palette to use. Default is None.	`None`
`curved_lines`	`bool`	If True, the plot will have curved instead of straight lines. Default is True.	`True`

Returns:

Type	Description
`Figure`	A matplotlib figure containing the parallel coordinates plot.

Raises:

Type	Description
`EmptyDataFrameException`	Raised when the DataFrame is empty.
`InvalidColumnException`	Raised when the color column is not found in the DataFrame.
`InconsistentDataTypesException`	Raised when the color column has multiple data types.

5.11 PCA

```
compute_pca(data, number_of_components, columns=None, scaler_type='standard', nodata_handling='remove', nodata=None)
```

Compute defined number of principal components for numeric input data.

Before computation, data is scaled according to specified scaler and NaN values removed or replaced. Optionally, a nodata value can be given to handle similarly as NaN values.

If input data is a Numpy array, interpretation of the data depends on its dimensions. If array is 3D, it is interpreted as a multiband raster/stacked rasters format (bands, rows, columns). If array is 2D, it is interpreted as table-like data, where each column represents a variable/raster band and each row a data point (similar to a Dataframe).

Parameters:

Name	Type	Description	Default
`data`	`Union[ndarray, DataFrame, GeoDataFrame]`	Input data for PCA.	required
`number_of_components`	`int`	The number of principal components to compute. Should be >= 1 and at most the number of numeric columns if input is (Geo)Dataframe.	required
`columns`	`Optional[Sequence[str]]`	Select columns used for the PCA. Other columns are excluded from PCA, but added back to the result Dataframe intact. Only relevant if input is (Geo)Dataframe. Defaults to None.	`None`
`scaler_type`	`Literal[standard, min_max, robust]`	Transform data according to a specified Sklearn scaler. Options are "standard", "min_max" and "robust". Defaults to "standard".	`'standard'`
`nodata_handling`	`Literal[remove, replace]`	If observations with nodata (NaN and given `nodata`) should be removed for the time of PCA computation or replaced with column/band mean. Defaults to "remove".	`'remove'`
`nodata`	`Optional[Number]`	Define a nodata value to remove. Defaults to None.	`None`

Returns:

Type	Description
`Union[ndarray, DataFrame, GeoDataFrame]`	The computed principal components in corresponding format as the input data and the
`ndarray`	explained variance ratios for each component.

Raises:

Type	Description
`EmptyDataException`	The input is empty.
`InvalidColumnException`	Selected columns are not found in the input Dataframe.
`InvalidNumberOfPrincipalComponents`	The number of principal components is less than 1 or more than number of columns if input was (Geo)DataFrame.
`InvalidParameterValueException`	If value for `number_of_components` is invalid.

`plot_pca(pca_df, explained_variances=None, color_column_name=None, save_path=None)`

Plot a scatter matrix of different principal component combinations.

Automatically filters columns that do not start with "principal_component" for plotting. This tool is designed to work smoothly on `compute_pca` outputs.

Parameters:

Name	Type	Description	Default
`pca_df`	`DataFrame`	A DataFrame containing computed principal components.	required
`explained_variances`	`Optional[ndarray]`	The explained variance ratios for each principal component. Used for labeling axes in the plot. Optional parameter. Defaults to None.	`None`
`color_column_name`	`Optional[str]`	Name of the column that will be used for color-coding data points. Typically a categorical variable in the original data. Optional parameter, no colors if not provided. Defaults to None.	`None`
`save_path`	`Optional[str]`	The save path for the plot. Optional parameter, no saving if not provided. Defaults to None.	`None`

Returns:

Type	Description
`PairGrid`	A Seaborn pairgrid containing the PCA scatter matrix.

Raises:

Type	Description
`InvalidColumnException`	DataFrame does not contain the given color column.

6. Prediction

6.1 Fuzzy overlay

`and_overlay(data)`

Compute an 'and' overlay operation with fuzzy logic.

Parameters:

Name	Type	Description	Default
`data`	`Union[Sequence[ndarray], ndarray]`	The input data as a series of 2D/3D Numpy arrays or as a 3D Numpy array. All found 2D arrays are overlayed. Input data should contain at least 2D Numpy arrays and data should be in the range [0, 1].	required

Returns:

Type	Description
`ndarray`	2D Numpy array with the result of the 'and' overlay operation. Values are in range [0, 1].

Raises:

Type	Description
`InvalidDatasetException`	If input data contains less than two 2D Numpy arrays/raster bands.
`InvalidParameterValueException`	If data values are not in range [0, 1].

`gamma_overlay(data, gamma=0.5)`

Compute a 'gamma' overlay operation with fuzzy logic.

Parameters:

Name	Type	Description	Default
`data`	`Union[Sequence[ndarray], ndarray]`	The input data as a series of 2D/3D Numpy arrays or as a 3D Numpy array. All found 2D arrays are overlayed. Input data should contain at least 2D Numpy arrays and data should be in the range [0, 1].	required
`gamma`	`float`	The gamma parameter. With gamma value of 0, the result will be the same as 'product' overlay. When gamma is closer to 1, the weight of the 'sum' overlay is increased. Defaults to 0.5. Value must be in the range [0, 1].	`0.5`

Returns:

Type	Description
`ndarray`	2D Numpy array with the result of the 'gamma' overlay operation. Values are in range [0, 1].

Raises:

Type	Description
`InvalidDatasetException`	If input data contains less than two 2D Numpy arrays/raster bands.
`InvalidParameterValueException`	If data values or gamma are not in range [0, 1].

`or_overlay(data)`

Compute an 'or' overlay operation with fuzzy logic.

Parameters:

Name	Type	Description	Default
`data`	`Union[Sequence[ndarray], ndarray]`	The input data as a series of 2D/3D Numpy arrays or as a 3D Numpy array. All found 2D arrays are overlayed. Input data should contain at least 2D Numpy arrays and data should be in the range [0, 1].	required

Returns:

Type	Description
`ndarray`	2D Numpy array with the result of the 'or' overlay operation. Values are in range [0, 1].

Raises:

Type	Description
`InvalidDatasetException`	If input data contains less than two 2D Numpy arrays/raster bands.
`InvalidParameterValueException`	If data values are not in range [0, 1].

`product_overlay(data)`

Compute a 'product' overlay operation with fuzzy logic.

Parameters:

Name	Type	Description	Default
`data`	`Union[Sequence[ndarray], ndarray]`	The input data as a series of 2D/3D Numpy arrays or as a 3D Numpy array. All found 2D arrays are overlayed. Input data should contain at least 2D Numpy arrays and data should be in the range [0, 1].	required

Returns:

Type	Description
`ndarray`	2D Numpy array with the result of the 'product' overlay operation. Values are in range [0, 1].

Raises:

Type	Description
`InvalidDatasetException`	If input data contains less than two 2D Numpy arrays/raster bands.
`InvalidParameterValueException`	If data values are not in range [0, 1].

`sum_overlay(data)`

Compute a 'sum' overlay operation with fuzzy logic.

Parameters:

Name	Type	Description	Default
`data`	`Union[Sequence[ndarray], ndarray]`	The input data as a series of 2D/3D Numpy arrays or as a 3D Numpy array. All found 2D arrays are overlayed. Input data should contain at least 2D Numpy arrays and data should be in the range [0, 1].	required

Returns:

Type	Description
`ndarray`	2D Numpy array with the result of the 'sum' overlay operation. Values are in range [0, 1].

Raises:

Type	Description
`InvalidDatasetException`	If input data contains less than two 2D Numpy arrays/raster bands.
`InvalidParameterValueException`	If data values are not in range [0, 1].

6.2 Gradient boosting

```
gradient_boosting_classifier_train(X, y, validation_method='split', metrics=['accuracy'], split_size=0.2, cv_folds=5, loss='log_loss', learning_rate=0.1,
n_estimators=100, max_depth=3, subsample=1.0, verbose=0, random_state=None, **kwargs)
```

Train and optionally validate a Gradient Boosting classifier model using Sklearn.

Various options and configurations for model performance evaluation are available. No validation, split to train and validation parts, and cross-validation can be chosen. If validation is performed, metric(s) to calculate can be defined and validation process configured (cross-validation method, number of folds, size of the split). Depending on the details of the validation process, the output metrics dictionary can be empty, one-dimensional or nested.

For more information about Sklearn Gradient Boosting classifier read the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

Parameters:

Name	Type	Description	Default
`X`	`Union[ndarray, DataFrame]`	Training data.	required
`y`	`Union[ndarray, Series]`	Target labels.	required
`validation_method`	`Literal[split, kfold_cv, skfold_cv, loo_cv, none]`	Validation method to use. "split" divides data into two parts, "kfold_cv" performs k-fold cross-validation, "skfold_cv" performs stratified k-fold cross-validation, "loo_cv" performs leave-one-out cross-validation and "none" will not validate model at all (in this case, all X and y will be used solely for training).	`'split'`
`metrics`	`Sequence[Literal[accuracy, precision, recall, f1, auc]]`	Metrics to use for scoring the model. Defaults to "accuracy".	`['accuracy']`
`split_size`	`float`	Fraction of the dataset to be used as validation data (rest is used for training). Used only when validation_method is "split". Defaults to 0.2.	`0.2`
`cv_folds`	`int`	Number of folds used in cross-validation. Used only when validation_method is "kfold_cv" or "skfold_cv". Defaults to 5.	`5`
`loss`	`Literal[log_loss, exponential]`	The loss function to be optimized. Defaults to "log_loss" (same as in logistic regression).	`'log_loss'`
`learning_rate`	`Number`	Shrinks the contribution of each tree. Values must be >= 0. Defaults to 0.1.	`0.1`
`n_estimators`	`int`	The number of boosting stages to run. Gradient boosting is fairly robust to over-fitting so a large number can result in better performance. Values must be >= 1. Defaults to 100.	`100`
`max_depth`	`Optional[int]`	Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Values must be >= 1 or None, in which case nodes are expanded until all leaves are pure or until all leaves contain less than	`3`

Name	Type	Description	Default
		min_samples_split samples. Defaults to 3.	
`subsample`	`Number`	The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. Subsample interacts with the parameter n_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias. Values must be in the range 0.0 < x <= 1.0. Defaults to 1.0.	`1.0`
`verbose`	`int`	Specifies if modeling progress and performance should be printed. 0 doesn't print, 1 prints once in a while depending on the number of tress, 2 or above will print for every tree.	`0`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`**kwargs`		Additional parameters for Sklearn's GradientBoostingClassifier.	`{}`

Returns:

Type	Description
`Tuple[GradientBoostingClassifier, dict]`	The trained GradientBoostingClassifier and metric scores as a dictionary.

Raises:

Type	Description
`InvalidParameterValueException`	If some of the numeric parameters are given invalid input values.
`NonMatchingParameterLengthsException`	X and y have mismatching sizes.

```
gradient_boosting_regressor_train(X, y, validation_method='split', metrics=['mse'], split_size=0.2, cv_folds=5, loss='squared_error', learning_rate=0.1, n_estimators=100, max_depth=3, subsample=1.0, verbose=0, random_state=None, **kwargs)
```

Train and optionally validate a Gradient Boosting regressor model using Sklearn.

Various options and configurations for model performance evaluation are available. No validation, split to train and validation parts, and cross-validation can be chosen. If validation is performed, metric(s) to calculate can be defined and validation process configured (cross-validation method, number of folds, size of the split). Depending on the details of the validation process, the output metrics dictionary can be empty, one-dimensional or nested.

For more information about Sklearn Gradient Boosting regressor read the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

Parameters:

Name	Type	Description	Default
`X`	`Union[ndarray, DataFrame]`	Training data.	required
`y`	`Union[ndarray, Series]`	Target labels.	required
`validation_method`	`Literal[split, kfold_cv, skfold_cv, loo_cv, none]`	Validation method to use. "split" divides data into two parts, "kfold_cv" performs k-fold cross-validation, "skfold_cv" performs stratified k-fold cross-validation, "loo_cv" performs leave-one-out cross-validation and "none" will not validate model at all (in this case, all X and y will be used solely for training).	`'split'`
`metrics`	`Sequence[Literal[mse, rmse, mae, r2]]`	Metrics to use for scoring the model. Defaults to "mse".	`['mse']`
`split_size`	`float`	Fraction of the dataset to be used as validation data (rest is used for training). Used only when validation_method is "split". Defaults to 0.2.	`0.2`
`cv_folds`	`int`	Number of folds used in cross-validation. Used only when validation_method is "kfold_cv" or "skfold_cv". Defaults to 5.	`5`
`loss`	`Literal[squared_error, absolute_error, huber, quantile]`	The loss function to be optimized. Defaults to "squared_error".	`'squared_error'`
`learning_rate`	`Number`	Shrinks the contribution of each tree. Values must be > 0. Defaults to 0.1.	`0.1`
`n_estimators`	`int`	The number of boosting stages to run. Gradient boosting is fairly robust to over-fitting so a large number can result in better performance. Values must be >= 1. Defaults to 100.	`100`
`max_depth`	`Optional[int]`	Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Values must be >= 1 or None, in which case nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Defaults to 3.	`3`

Name	Type	Description	Default
`subsample`	`Number`	The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. Subsample interacts with the parameter n_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias. Values must be in the range 0.0 < x <= 1.0. Defaults to 1.	`1.0`
`verbose`	`int`	Specifies if modeling progress and performance should be printed. 0 doesn't print, 1 prints once in a while depending on the number of tress, 2 or above will print for every tree.	`0`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`**kwargs`		Additional parameters for Sklearn's GradientBoostingRegressor.	`{}`

Returns:

Type	Description
`Tuple[GradientBoostingRegressor, dict]`	The trained GradientBoostingRegressor and metric scores as a dictionary.

Raises:

Type	Description
`InvalidParameterValueException`	If some of the numeric parameters are given invalid input values.
`NonMatchingParameterLengthsException`	X and y have mismatching sizes.

6.3 Logistic regression

`logistic_regression_train(X, y, validation_method='split', metrics=['accuracy'], split_size=0.2, cv_folds=5, penalty='l2', max_iter=100, solver='lbfgs', verbose=0, random_state=None, **kwargs)`

Train and optionally validate a Logistic Regression classifier model using Sklearn.

Various options and configurations for model performance evaluation are available. No validation, split to train and validation parts, and cross-validation can be chosen. If validation is performed, metric(s) to calculate can be defined and validation process configured (cross-validation method, number of folds, size of the split). Depending on the details of the validation process, the output metrics dictionary can be empty, one-dimensional or nested.

The choice of the algorithm depends on the penalty chosen. Supported penalties by solver: 'lbfgs' - ['l2', None] 'liblinear' - ['l1', 'l2'] 'newton-cg' - ['l2', None] 'newton-cholesky' - ['l2', None] 'sag' - ['l2', None] 'saga' - ['elasticnet', 'l1', 'l2', None]

For more information about Sklearn Logistic Regression, read the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Parameters:

Name	Type	Description	Default
`X`	`Union[ndarray, DataFrame]`	Training data.	required
`y`	`Union[ndarray, Series]`	Target labels.	required
`validation_method`	`Literal[split, kfold_cv, skfold_cv, loo_cv, none]`	Validation method to use. "split" divides data into two parts, "kfold_cv" performs k-fold cross-validation, "skfold_cv" performs stratified k-fold cross-validation, "loo_cv" performs leave-one-out cross-validation and "none" will not validate model at all (in this case, all X and y will be used solely for training).	`'split'`
`metrics`	`Sequence[Literal[accuracy, precision, recall, f1, auc]]`	Metrics to use for scoring the model. Defaults to "accuracy".	`['accuracy']`
`split_size`	`float`	Fraction of the dataset to be used as validation data (rest is used for training). Used only when validation_method is "split". Defaults to 0.2.	`0.2`
`cv_folds`	`int`	Number of folds used in cross-validation. Used only when validation_method is "kfold_cv" or "skfold_cv". Defaults to 5.	`5`
`penalty`	`Literal[l1, l2, elasicnet, None]`	Specifies the norm of the penalty. Defaults to 'l2'.	`'l2'`
`max_iter`	`int`	Maximum number of iterations taken for the solvers to converge. Defaults to 100.	`100`
`solver`	`Literal[lbfgs, liblinear, newton - cg, newton - cholesky, sag, saga]`	Algorithm to use in the optimization problem. Defaults to 'lbfgs'.	`'lbfgs'`
`verbose`	`int`	Specifies if modeling progress and performance should be printed. 0 doesn't print, values 1 or above will produce prints.	`0`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`**kwargs`		Additional parameters for Sklearn's LogisticRegression.	`{}`

Returns:

Type	Description
`Tuple[LogisticRegression, dict]`	The trained Logistric Regression classifier and metric scores as a dictionary.

Raises:

Type	Description
`InvalidParameterValueException`	If some of the numeric parameters are given invalid input values.
`NonMatchingParameterLengthsException`	X and y have mismatching sizes.

6.4 Logistic regression

`load_model(path)`

Load a Sklearn model from a .joblib file.

Parameters:

Name	Type	Description	Default
`path`	`Path`	Path from where the model should be loaded. Include the .joblib file extension.	required

Returns:

Type	Description
`BaseEstimator`	Loaded model.

`prepare_data_for_ml(feature_raster_files, label_file=None)`

Prepare data ready for machine learning model training.

Performs the following steps: - Read all bands of all feature/evidence rasters into a stacked Numpy array - Read label data (and rasterize if a vector file is given) - Create a nodata mask using all feature rasters and labels, and mask nodata cells out

Parameters:

Name	Type	Description	Default
`feature_raster_files`	`Sequence[Union[str, PathLike]]`	List of filepaths of feature/evidence rasters. Files should only include raster that have the same grid properties and extent.	required
`label_file`	`Optional[Union[str, PathLike]]`	Filepath to label (deposits) data. File can be either a vector file or raster file. If a vector file is provided, it will be rasterized into similar grid as feature rasters. If a raster file is provided, it needs to have same grid properties and extent as feature rasters. Optional parameter and can be omitted if preparing data for predicting. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Feature data (X) in prepared shape.
`Optional[ndarray]`	Target labels (y) in prepared shape (if `label_file` was given).
`Profile`	Refrence raster metadata .
`Any`	Nodata mask applied to X and y.

Raises:

Type	Description
`NonMatchingRasterMetadataException`	Input feature rasters don't have same grid properties.

`reshape_predictions(predictions, height, width, nodata_mask=None)`

Reshape 1D prediction ouputs into 2D Numpy array.

The output is ready to be visualized and saved as a raster.

Parameters:

Name	Type	Description	Default
`predictions`	`ndarray`	A 1D Numpy array with raw prediction data from `predict` function.	required
`height`	`int`	Height of the output array	required
`width`	`int`	Width of the output array	required
`nodata_mask`	`Optional[ndarray]`	Nodata mask used to reconstruct original shape of data. This is the same mask applied to data before predicting to remove nodata. If any nodata was removed before predicting, this mask is required to reconstruct the original shape of data. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Predictions as a 2D Numpy array in the original array shape.

`save_model(model, path)`

Save a trained Sklearn model to a .joblib file.

Parameters:

Name	Type	Description	Default
`model`	`BaseEstimator`	Trained model.	required
`path`	`Path`	Path where the model should be saved. Include the .joblib file extension.	required

`split_data(*data, split_size=0.2, random_state=None, shuffle=True)`

Split data into two parts. Can be used for train-test or train-validation splits.

For more guidance, read documentation of sklearn.model_selection.train_test_split: (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html).

Parameters:

Name	Type	Description	Default
`*data`	`Union[ndarray, DataFrame, csr_matrix, List[Number]]`	Data to be split. Multiple datasets can be given as input (for example X and y), but they need to have the same length. All datasets are split into two and the parts returned (for example X_train, X_test, y_train, y_test).	`()`
`split_size`	`float`	The proportion of the second part of the split. Typically this is the size of test/validation part. The first part will be complemental proportion. For example, if split_size = 0.2, the first part will have 80% of the data and the second part 20% of the data. Defaults to 0.2.	`0.2`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`shuffle`	`bool`	If data is shuffled before splitting. Defaults to True.	`True`

Returns:

Type	Description
`List[Union[ndarray, DataFrame, csr_matrix, List[Number]]]`	List containing splits of inputs (two outputs per input).

6.5 MLP

```
train_MLP_classifier(X, y, neurons, validation_split=0.2, validation_data=None, activation='relu', output_neurons=1, last_activation='sigmoid', epochs=50, batch_size=32, optimizer='adam', learning_rate=0.001, loss_function='binary_crossentropy', dropout_rate=None, early_stopping=True, es_patience=5, metrics=['accuracy'], random_state=None)
```

Train MLP (Multilayer Perceptron) using Keras.

Creates a Sequential model with Dense NN layers. For each element in `neurons`, Dense layer with corresponding dimensionality/neurons is created with the specified activation function (`activation`). If `dropout_rate` is specified, a Dropout layer is added after each Dense layer.

Parameters default to a binary classification model using sigmoid as last activation, binary crossentropy as loss function and 1 output neuron/unit.

For more information about Keras models, read the documentation here: https://keras.io/.

Parameters:

Name	Type	Description	Default
`X`	`ndarray`	Input data. Should be a 2-dimensional array where each row represents a sample and each column a feature. Features should ideally be normalized or standardized.	required
`y`	`ndarray`	Target labels. For binary classification, y should be a 1-dimensional array of binary labels (0 or 1). For multi-class classification, y should be a 2D array with one-hot encoded labels. The number of columns should match the number of classes.	required
`neurons`	`Sequence[int]`	Number of neurons in each hidden layer.	required
`validation_split`	`Optional[float]`	Fraction of data used for validation during training. Value must be > 0 and < 1 or None. Defaults to 0.2.	`0.2`
`validation_data`	`Optional[Tuple[ndarray, ndarray]]`	Separate dataset used for validation during training. Overrides validation_split if provided. Expected data form is (X_valid, y_valid). Defaults to None.	`None`
`activation`	`Literal[relu, linear, sigmoid, tanh]`	Activation function used in each hidden layer. Defaults to 'relu'.	`'relu'`
`output_neurons`	`int`	Number of neurons in the output layer. Defaults to 1.	`1`
`last_activation`	`Literal[sigmoid, softmax]`	Activation function used in the output layer. Defaults to 'sigmoid'.	`'sigmoid'`
`epochs`	`int`	Number of epochs to train the model. Defaults to 50.	`50`

Name	Type	Description	Default
`batch_size`	`int`	Number of samples per gradient update. Defaults to 32.	`32`
`optimizer`	`Literal[adam, adagrad, rmsprop, sdg]`	Optimizer to be used. Defaults to 'adam'.	`'adam'`
`learning_rate`	`Number`	Learning rate to be used in training. Value must be > 0. Defalts to 0.001.	`0.001`
`loss_function`	`Literal[binary_crossentropy, categorical_crossentropy]`	Loss function to be used. Defaults to 'binary_crossentropy'.	`'binary_crossentropy'`
`dropout_rate`	`Optional[Number]`	Fraction of the input units to drop. Value must be >= 0 and <= 1. Defaults to None.	`None`
`early_stopping`	`bool`	Whether or not to use early stopping in training. Defaults to True.	`True`
`es_patience`	`int`	Number of epochs with no improvement after which training will be stopped. Defaults to 5.	`5`
`metrics`	`Optional[Sequence[Literal[accuracy, precision, recall, f1_score]]]`	Metrics to be evaluated by the model during training and testing. Defaults to ['accuracy'].	`['accuracy']`
`random_state`	`Optional[int]`	Seed for random number generation. Sets Python, Numpy and Tensorflow seeds to make program deterministic. Defaults to None (random state / seed).	`None`

Returns:

Type	Description
`Tuple[Model, dict]`	Trained MLP model and training history.

Raises:

Type	Description
`InvalidParameterValueException`	Some of the numeric parameters have invalid values.
`InvalidDataShapeException`	Shape of X or y is invalid.

```
train_MLP_regressor(X, y, neurons, validation_split=0.2, validation_data=None, activation='relu', output_neurons=1, last_activation='linear', epochs=50, batch_size=32, optimizer='adam', learning_rate=0.001, loss_function='mse', dropout_rate=None, early_stopping=True, es_patience=5, metrics=['mse'], random_state=None)
```

Train MLP (Multilayer Perceptron) using Keras.

Creates a Sequential model with Dense NN layers. For each element in `neurons`, Dense layer with corresponding dimensionality/ neurons is created with the specified activation function (`activation`). If `dropout_rate` is specified, a Dropout layer is added after each Dense layer.

For more information about Keras models, read the documentation here: https://keras.io/.

Parameters:

Name	Type	Description	Default
`X`	`ndarray`	Input data. Should be a 2-dimensional array where each row represents a sample and each column a feature. Features should ideally be normalized or standardized.	required
`y`	`ndarray`	Target labels. Should be a 1-dimensional array where each entry corresponds to the continuous target value for the respective sample in X.	required
`neurons`	`Sequence[int]`	Number of neurons in each hidden layer.	required
`validation_split`	`Optional[float]`	Fraction of data used for validation during training. Value must be > 0 and < 1 or None. Defaults to 0.2.	`0.2`
`validation_data`	`Optional[Tuple[ndarray, ndarray]]`	Separate dataset used for validation during training. Overrides validation_split if provided. Expected data form is (X_valid, y_valid). Defaults to None.	`None`
`activation`	`Literal[relu, linear, sigmoid, tanh]`	Activation function used in each hidden layer. Defaults to 'relu'.	`'relu'`
`output_neurons`	`int`	Number of neurons in the output layer. Defaults to 1.	`1`
`last_activation`	`Literal[linear]`	Activation function used in the output layer. Defaults to 'linear'.	`'linear'`
`epochs`	`int`	Number of epochs to train the model. Defaults to 50.	`50`
`batch_size`	`int`	Number of samples per gradient update. Defaults to 32.	`32`
`optimizer`	`Literal[adam, adagrad, rmsprop, sdg]`	Optimizer to be used. Defaults to 'adam'.	`'adam'`
`learning_rate`	`Number`	Learning rate to be used in training. Value must be > 0. Defalts to 0.001.	`0.001`
`loss_function`	`Literal[mse, mae, hinge, huber]`	Loss function to be used. Defaults to 'mse'.	`'mse'`
`dropout_rate`	`Optional[Number]`	Fraction of the input units to drop. Value must be >= 0 and <= 1. Defaults to None.	`None`
`early_stopping`	`bool`	Whether or not to use early stopping in training. Defaults to True.	`True`
`es_patience`	`int`	Number of epochs with no improvement after which training will be stopped. Defaults to 5.	`5`
`metrics`	`Optional[Sequence[Literal[mse, rmse, mae]]]`	Metrics to be evaluated by the model during training and testing. Defaults to ['mse'].	`['mse']`
`random_state`	`Optional[int]`		`None`

Name	Type	Description	Default
		Seed for random number generation. Sets Python, Numpy and Tensorflow seeds to make program deterministic. Defaults to None (random state / seed).	

Returns:

Type	Description
`Tuple[Model, dict]`	Trained MLP model and training history.

Raises:

Type	Description
`InvalidParameterValueException`	Some of the numeric parameters have invalid values.
`InvalidDataShapeException`	Shape of X or y is invalid.

6.6 Random forests

```
random_forest_classifier_train(X, y, validation_method='split', metrics=['accuracy'], split_size=0.2, cv_folds=5, n_estimators=100, criterion='gini',
max_depth=None, verbose=0, random_state=None, **kwargs)
```

Train and optionally validate a Random Forest classifier model using Sklearn.

Various options and configurations for model performance evaluation are available. No validation, split to train and validation parts, and cross-validation can be chosen. If validation is performed, metric(s) to calculate can be defined and validation process configured (cross-validation method, number of folds, size of the split). Depending on the details of the validation process, the output metrics dictionary can be empty, one-dimensional or nested.

For more information about Sklearn Random Forest classifier, read the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Parameters:

Name	Type	Description	Default
`X`	`Union[ndarray, DataFrame]`	Training data.	required
`y`	`Union[ndarray, Series]`	Target labels.	required
`validation_method`	`Literal[split, kfold_cv, skfold_cv, loo_cv, none]`	Validation method to use. "split" divides data into two parts, "kfold_cv" performs k-fold cross-validation, "skfold_cv" performs stratified k-fold cross-validation, "loo_cv" performs leave-one-out cross-validation and "none" will not validate model at all (in this case, all X and y will be used solely for training).	`'split'`
`metrics`	`Sequence[Literal[accuracy, precision, recall, f1]]`	Metrics to use for scoring the model. Defaults to "accuracy".	`['accuracy']`
`split_size`	`float`	Fraction of the dataset to be used as validation data (rest is used for training). Used only when validation_method is "split". Defaults to 0.2.	`0.2`
`cv_folds`	`int`	Number of folds used in cross-validation. Used only when validation_method is "kfold_cv" or "skfold_cv". Defaults to 5.	`5`
`n_estimators`	`int`	The number of trees in the forest. Defaults to 100.	`100`
`criterion`	`Literal[gini, entropy, log_loss]`	The function to measure the quality of a split. Defaults to "gini".	`'gini'`
`max_depth`	`Optional[int]`	The maximum depth of the tree. Values must be >= 1 or None, in which case nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Defaults to None.	`None`
`verbose`	`int`	Specifies if modeling progress and performance should be printed. 0 doesn't print, values 1 or above will produce prints.	`0`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`**kwargs`		Additional parameters for Sklearn's RandomForestClassifier.	`{}`

Returns:

Type	Description
`Tuple[RandomForestClassifier, dict]`	The trained RandomForestClassifier and metric scores as a dictionary.

Raises:

Type	Description
`InvalidParameterValueException`	If some of the numeric parameters are given invalid input values.
`NonMatchingParameterLengthsException`	X and y have mismatching sizes.

```
random_forest_regressor_train(X, y, validation_method='split', metrics=['mse'], split_size=0.2, cv_folds=5, n_estimators=100, criterion='squared_error', max_depth=None, verbose=0, random_state=None, **kwargs)
```

Train and optionally validate a Random Forest regressor model using Sklearn.

Various options and configurations for model performance evaluation are available. No validation, split to train and validation parts, and cross-validation can be chosen. If validation is performed, metric(s) to calculate can be defined and validation process configured (cross-validation method, number of folds, size of the split). Depending on the details of the validation process, the output metrics dictionary can be empty, one-dimensional or nested.

For more information about Sklearn Random Forest regressor, read the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

Parameters:

Name	Type	Description	Default
`X`	`Union[ndarray, DataFrame]`	Training data.	required
`y`	`Union[ndarray, Series]`	Target labels.	required
`validation_method`	`Literal[split, kfold_cv, skfold_cv, loo_cv, none]`	Validation method to use. "split" divides data into two parts, "kfold_cv" performs k-fold cross-validation, "skfold_cv" performs stratified k-fold cross-validation, "loo_cv" performs leave-one-out cross-validation and "none" will not validate model at all (in this case, all X and y will be used solely for training).	`'split'`
`metrics`	`Sequence[Literal[mse, rmse, mae, r2]]`	Metrics to use for scoring the model. Defaults to "mse".	`['mse']`
`split_size`	`float`	Fraction of the dataset to be used as validation data (rest is used for training). Used only when validation_method is "split". Defaults to 0.2.	`0.2`
`cv_folds`	`int`	Number of folds used in cross-validation. Used only when validation_method is "kfold_cv" or "skfold_cv". Defaults to 5.	`5`
`n_estimators`	`int`	The number of trees in the forest. Defaults to 100.	`100`
`criterion`	`Literal[squared_error, absolute_error, friedman_mse, poisson]`	The function to measure the quality of a split. "absolute_error" results in significantly longer training time than "squared_error". Defaults to "squared_error".	`'squared_error'`
`max_depth`	`Optional[int]`	The maximum depth of the tree. Values must be >= 1 or None, in which case nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Defaults to None.	`None`

Name	Type	Description	Default
`verbose`	`int`	Specifies if modeling progress and performance should be printed. 0 doesn't print, values 1 or above will produce prints.	`0`
`random_state`	`Optional[int]`	Seed for random number generation. Defaults to None.	`None`
`**kwargs`		Additional parameters for Sklearn's RandomForestRegressor.	`{}`

Returns:

Type	Description
`Tuple[RandomForestRegressor, dict]`	The trained RandomForestRegressor and metric scores as a dictionary.

Raises:

Type	Description
`InvalidParameterValueException`	If some of the numeric parameters are given invalid input values.
`NonMatchingParameterLengthsException`	X and y have mismatching sizes.

6.7 Weights of evidence

`weights_of_evidence_calculate_responses(output_arrays, nr_of_deposits, nr_of_pixels)`

Calculate the posterior probabilities for the given generalized weight arrays.

Parameters:

Name	Type	Description	Default
`output_arrays`	`Sequence[Dict[str, ndarray]]`	List of output array dictionaries returned by weights of evidence calculations. For each dictionary, generalized weight and generalized standard deviation arrays are used and summed together pixel-wise to calculate the posterior probabilities. If generalized arrays are not found, the W+ and S_W+ arrays are used (so if outputs from unique weight calculations are used for this function).	required
`nr_of_deposits`	`int`	Number of deposit pixels in the input data for weights of evidence calculations.	required
`nr_of_pixels`	`int`	Number of evidence pixels in the input data for weights of evidence calculations.	required

Returns:

Type	Description
`ndarray`	Array of posterior probabilites.
`ndarray`	Array of standard deviations in the posterior probability calculations.
`ndarray`	Array of confidence of the prospectivity values obtained in the posterior probability array.

`weights_of_evidence_calculate_weights(evidential_raster, deposits, raster_nodata=None, weights_type='unique', studentized_contrast_threshold=1, arrays_to_generate=None)`

Calculate weights of spatial associations.

Parameters:

Name	**Type**	**Description**	**Default**
`evidential_raster`	`DatasetReader`	The evidential raster.	required
`deposits`	`GeoDataFrame`	Vector data representing the mineral deposits or occurences point data.	required
`raster_nodata`	`Optional[Number]`	If nodata value of raster is wanted to specify manually. Optional parameter, defaults to None (nodata from raster metadata is used).	`None`
`weights_type`	`Literal[unique, categorical, ascending, descending]`	Accepted values are 'unique', 'categorical', 'ascending' and 'descending'. Unique weights does not create generalized classes and does not use a studentized contrast threshold value while categorical, cumulative ascending and cumulative descending do. Categorical weights are calculated so that all classes with studentized contrast below the defined threshold are grouped into one generalized class. Cumulative ascending and descending weights find the class with max contrast and group classes above/ below into generalized classes. Generalized weights are also calculated for generalized classes.	`'unique'`
`studentized_contrast_threshold`	`Number`	Studentized contrast threshold value used with 'categorical', 'ascending' and 'descending' weight types. Used either as reclassification threshold directly (categorical) or to check that class with	`1`

Name	Type	Description	Default
		max contrast has studentized contrast value at least the defined value (cumulative). Defaults to 1.	
`arrays_to_generate`	`Optional[Sequence[str]]`	Arrays to generate from the computed weight metrics. All column names in the produced weights_df are valid choices. Defaults to ["Class", "W+", "S_W+] for "unique" weights_type and ["Class", "W+", "S_W+", "Generalized W+", "Generalized S_W+"] for the cumulative weight types.	`None`

Returns:

Type	Description
`DataFrame`	Dataframe with weights of spatial association between the input data.
`dict`	Dictionary of arrays for specified metrics.
`dict`	Raster metadata.
`int`	Number of deposit pixels.
`int`	Number of all evidence pixels.

Raises:

Type	Description
`ClassificationFailedException`	Unable to create generalized classes with the given studentized_contrast_threshold.
`InvalidColumnException`	Arrays to generate contains invalid column name(s).
`InvalidParameterValueException`	Input weights_type is not one of the accepted values.

7. Raster processing

7.1 Clipping

`clip_raster(raster, geodataframe)`

Clips a raster with polygon geometries.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The raster to be clipped.	required
`geodataframe`	`GeoDataFrame`	A geodataframe containing the geometries to do the clipping with. Should contain only polygon features.	required

Returns:

Type	Description
`ndarray`	The clipped raster data.
`dict`	The updated metadata.

Raises:

Type	Description
`NonMatchingCrsException`	The raster and geodataframe are not in the same CRS.
`GeometryTypeException`	The input geometries contain non-polygon features.

7.2 Create constant raster

```
create_constant_raster(constant_value, template_raster=None, coord_west=None, coord_north=None, coord_east=None, coord_south=None, target_epsg=None, target_pixel_size=None, raster_width=None, raster_height=None, nodata_value=None)
```

Create a constant raster based on a user-defined value.

Provide 3 methods for raster creation: 1. Set extent and coordinate system based on a template raster. 2. Set extent from origin, based on the western and northern coordinates and the pixel size. 3. Set extent from bounds, based on western, northern, eastern and southern points.

Always provide values for height and width for the last two options, which correspond to the desired number of pixels for rows and columns.

Parameters:

Name	Type	Description	Default
`constant_value`	`Number`	The constant value to use in the raster.	required
`template_raster`	`Optional[DatasetReader]`	An optional raster to use as a template for the output.	`None`
`coord_west`	`Optional[Number]`	The western coordinate of the output raster in [m].	`None`
`coord_east`	`Optional[Number]`	The eastern coordinate of the output raster in [m].	`None`
`coord_south`	`Optional[Number]`	The southern coordinate of the output raster in [m].	`None`
`coord_north`	`Optional[Number]`	The northern coordinate of the output raster in [m].	`None`
`target_epsg`	`Optional[int]`	The EPSG code for the output raster.	`None`
`target_pixel_size`	`Optional[int]`	The pixel size of the output raster.	`None`
`raster_width`	`Optional[int]`	The width of the output raster.	`None`
`raster_height`	`Optional[int]`	The height of the output raster.	`None`
`nodata_value`	`Optional[Number]`	The nodata value of the output raster.	`None`

Returns:

Type	Description
`Tuple[ndarray, dict]`	A tuple containing the output raster as a NumPy array and updated metadata.

Raises:

Type	Description
`InvalidParameterValueException`	Provide invalid input parameter.

7.3 Distance to anomaly

`distance_to_anomaly(anomaly_raster_profile, anomaly_raster_data, threshold_criteria_value, threshold_criteria)`

Calculate distance from raster cell to nearest anomaly.

The criteria for what is anomalous can be defined as a single number and criteria text of "higher" or "lower". Alternatively, the definition can be a range where values inside (criteria text of "within") or outside are marked as anomalous (criteria text of "outside"). If anomaly_raster_profile does contain "nodata" key, np.nan is assumed to correspond to nodata values.

Parameters:

Name	Type	Description	Default
`anomaly_raster_profile`	`Union[Profile, dict]`	The raster profile in which the distances to the nearest anomalous value are determined.	required
`anomaly_raster_data`	`ndarray`	The raster data in which the distances to the nearest anomalous value are determined.	required
`threshold_criteria_value`	`Union[Tuple[Number, Number], Number]`	Value(s) used to define anomalous. If the threshold criteria requires a tuple of values, the first value should be the minimum and the second the maximum value.	required
`threshold_criteria`	`Literal[lower, higher, in_between, outside]`	Method to define anomalous.	required

Returns:

Type	Description
`ndarray`	A 2D numpy array with the distances to anomalies computed
`Union[Profile, dict]`	and the original anomaly raster profile.

`distance_to_anomaly_gdal(anomaly_raster_profile, anomaly_raster_data, threshold_criteria_value, threshold_criteria, output_path, verbose=False)`

Calculate distance from raster cell to nearest anomaly.

Distance is calculated for each cell in the anomaly raster and saved to a new raster at output_path. The criteria for what is anomalous can be defined as a single number and criteria text of "higher" or "lower". Alternatively, the definition can be a range where values inside (criteria text of "within") or outside are marked as anomalous (criteria text of "outside"). If anomaly_raster_profile does contain "nodata" key, np.nan is assumed to correspond to nodata values.

Does not work on Windows.

Parameters:

Name	Type	Description	Default
`anomaly_raster_profile`	`Union[Profile, dict]`	The raster profile in which the distances to the nearest anomalous value are determined.	required
`anomaly_raster_data`	`ndarray`	The raster data in which the distances to the nearest anomalous value are determined.	required
`threshold_criteria_value`	`Union[Tuple[Number, Number], Number]`	Value(s) used to define anomalous.	required
`threshold_criteria`	`Literal[lower, higher, in_between, outside]`	Method to define anomalous.	required
`output_path`	`Path`	The path to the raster with the distances to anomalies calculated.	required
`verbose`	`bool`	Whether to print gdal_proximity output.	`False`

Returns:

Type	Description
`Path`	The path to the raster with the distances to anomalies calculated.

7.4 Extract values from raster

`extract_values_from_raster(raster_list, geodataframe, raster_column_names=None)`

Extract raster values using point data to a DataFrame.

If custom column names are not given, column names are file_name for singleband files and file_name_bandnumber for multiband files. If custom column names are given, there should be column names for each raster provided in the raster list.

Parameters:

Name	Type	Description	Default
`raster_list`	`Sequence[DatasetReader]`	List to extract values from.	required
`geodataframe`	`GeoDataFrame`	Object to extract values with.	required
`raster_column_names`	`Optional[Sequence[str]]`	List of optional column names for bands.	`None`

Returns:

Type	Description
`DataFrame`	Dataframe with x & y coordinates and the values from the raster file(s) as columns.

Raises:

Type	Description
`NonMatchingParameterLengthsException`	raster_list and raster_columns_names have different lengths.

7.5 Reclassify raster

`reclassify_with_defined_intervals(raster, interval_size, bands=None)`

Classify raster with defined intervals.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`interval_size`	`int`	The number of units in each interval.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with defined intervals.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Interval size is less than 1.

`reclassify_with_equal_intervals(raster, number_of_intervals, bands=None)`

Classify raster with equal intervals.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`number_of_intervals`	`int`	The number of intervals.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with equal intervals.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Number of intervals is less than 2.

`reclassify_with_geometrical_intervals(raster, number_of_classes, bands=None)`

Classify raster with geometrical intervals.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`number_of_classes`	`int`	The number of classes. The true number of classes is at most double the amount, depending how symmetrical the input data is.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with geometrical intervals.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Number of classes is less than 2.

`reclassify_with_manual_breaks(raster, breaks, bands=None)`

Classify raster with manual breaks.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`breaks`	`Sequence[int]`	List of break values for the classification.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with manual breaks.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.

`reclassify_with_natural_breaks(raster, number_of_classes, bands=None)`

Classify raster with natural breaks (Jenks Caspall).

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`number_of_classes`	`int`	The number of classes.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with natural breaks (Jenks Caspall).
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Number of classes is less than 2.

`reclassify_with_quantiles(raster, number_of_quantiles, bands=None)`

Classify raster with quantiles.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`number_of_quantiles`	`int`	The number of quantiles.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with quantiles.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Number of quantiles is less than 2.

`reclassify_with_standard_deviation(raster, number_of_intervals, bands=None)`

Classify raster with standard deviation.

If bands are not given, all bands are used for classification.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Raster to be classified.	required
`number_of_intervals`	`int`	The number of intervals.	required
`bands`	`Optional[Sequence[int]]`	Selected bands from multiband raster. Indexing begins from one. Defaults to None.	`None`

Returns:

Type	Description
`ndarray`	Raster data classified with standard deviation.
`dict`	Raster metadata.

Raises:

Type	Description
`InvalidRasterBandException`	All selected bands are not contained in the input raster.
`InvalidParameterValueException`	Number of intervals is less than 2.

7.6 Reprojecting

`reproject_raster(raster, target_crs, resampling_method='nearest')`

Reprojects raster to match given coordinate reference system (EPSG).

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The raster to be reprojected.	required
`target_crs`	`int`	Target CRS as EPSG code.	required
`resampling_method`	`Literal[nearest, bilinear, cubic, average, gauss, max, min]`	Resampling method. Most suitable method depends on the dataset and context. Nearest, bilinear and cubic are some common choices. This parameter defaults to nearest.	`'nearest'`

Returns:

Type	Description
`ndarray`	The reprojected raster data.
`dict`	The updated metadata.

Raises:

Type	Description
`NonMatchinCrsException`	Raster is already in the target CRS.

7.7 Resampling

`resample(raster, resolution, resampling_method='bilinear')`

Resamples raster according to given resolution.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The raster to be resampled.	required
`resolution`	`Number`	Target resolution i.e. cell size of the output raster.	required
`resampling_method`	`Literal[nearest, bilinear, cubic, average, gauss, max, min]`	Resampling method. Most suitable method depends on the dataset and context. Nearest, bilinear and cubic are some common choices. This parameter defaults to bilinear.	`'bilinear'`

Returns:

Type	Description
`ndarray`	The resampled raster data.
`dict`	The updated metadata.

Raises:

Type	Description
`NumericValueSignException`	Resolution is not a positive value.

7.8 Snapping

`snap_with_raster(raster, snap_raster)`

Snaps/aligns raster to given snap raster.

Raster is snapped from its left-bottom corner to nearest snap raster grid corner in left-bottom direction. If rasters are aligned, simply returns input raster data and metadata.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The raster to be clipped.	required
`snap_raster`	`DatasetReader`	The snap raster i.e. reference grid raster.	required

Returns:

Type	Description
`ndarray`	The snapped raster data.
`dict`	The updated metadata.

Raises:

Type	Description
`NonMatchingCrsException`	Raster and and snap raster are not in the same CRS.
`MatchingRasterGridException`	Raster grids are already aligned.

7.9 Unifying

```
unify_raster_grids(base_raster, rasters_to_unify, resampling_method='nearest', same_extent=False)
```

Unifies (reprojects, resamples, aligns and optionally clips) given rasters relative to base raster.

Parameters:

Name	Type	Description	Default
`base_raster`	`DatasetReader`	The base raster to determine target raster grid properties.	required
`rasters_to_unify`	`Sequence[DatasetReader]`	Rasters to be unified with the base raster.	required
`resampling_method`	`Literal[nearest, bilinear, cubic, average, gauss, max, min]`	Resampling method. Most suitable method depends on the dataset and context. Nearest, bilinear and cubic are some common choices. This parameter defaults to nearest.	`'nearest'`
`same_extent`	`bool`	If the unified rasters will be forced to have the same extent/bounds as the base raster. Expands smaller rasters with nodata cells. Defaults to False.	`False`

Returns:

Type	Description
`List[Tuple[ndarray, dict]]`	List of unified rasters' data and metadata. First element is the base raster.

Raises:

Type	Description
`InvalidParameterValueException`	Rasters to unify is empty.

7.10 Unique combinations in rasters

`unique_combinations(raster_list)`

Get combinations of raster values between rasters.

All bands in all rasters are used for analysis. The first band of the first raster is used for reference when making the output.

Parameters:

Name	Type	Description	Default
`raster_list`	`Sequence[DatasetReader]`	Rasters to be used for finding combinations.	required

Returns:

Type	Description
`ndarray`	Combinations of rasters.
`dict`	The metadata of the first raster in raster_list.

Raises:

Type	Description
`InvalidParameterValueException`	Input rasters don't have enough bands to perform the operation or input rasters are of different shape.

7.11 Windowing

`extract_window(raster, center_coords, height, width)`

Extract window from raster.

Center coordinate must be inside the raster but window can extent outside the raster in which case padding with raster nodata value is used. Args: raster: Source raster. center_coords: Center coordinates for window in form (x, y). The coordinates should be in the raster's CRS. height: Window height in pixels. width: Window width in pixels.

Returns:

Type	Description
`ndarray`	The extracted raster window.
`dict`	The updated metadata.

Raises:

Type	Description
`InvalidParameterValueException`	Window size is too small.
`CoordinatesOutOfBoundException`	Window center coordinates are out of raster bounds.

7.12 Derivatives

7.12.1 Classification

```
CLASSIFY_ASPECT(RASTER, UNIT='RADIANS', NUM_CLASSES=8)
```

Classify an aspect raster data set.

Can classify an aspect raster into 8 or 16 equally spaced directions with intervals of pi/4 and pi/8, respectively.

Exemplary for 8 classes, the center of the intervall for North direction is 0°/360° and edges are [337.5°, 22.5°], counting forward in clockwise direction. For 16 classes, the intervall-width is half with edges at [348,75°, 11,25°].

Directions and interval for 8 classes: N: (337.5, 22.5), NE: (22.5, 67.5), E: (67.5, 112.5), SE: (112.5, 157.5), S: (157.5, 202.5), SW: (202.5, 247.5), W: (247.5, 292.5), NW: (292.5, 337.5)

Directions and interval for 16 classes: N: (348.75, 11.25), NNE: (11.25, 33.75), NE: (33.75, 56.25), ENE: (56.25, 78.75), E: (78.75, 101.25), ESE: (101.25, 123.75), SE: (123.75, 146.25), SSE: (146.25, 168.75), S: (168.75, 191.25), SSW: (191.25, 213.75), SW: (213.75, 236.25), WSW: (236.25, 258.75), W: (258.75, 281.25), WNW: (281.25, 303.75), NW: (303.75, 326.25), NNW: (326.25, 348.75)

Flat pixels (input value: -1) will be kept, the class is called ND (not defined).

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster data.	required
`unit`	`Literal[radians, degrees]`	The unit of the input raster. Either "degrees" or "radians"	`'radians'`
`num_classes`	`int`	The number of classes for discretization. Either 8 or 16 classes allowed.	`8`

Returns:

Type	Description
`tuple[ndarray, dict, dict]`	The classified aspect raster, a class mapping dictionary and the updated metadata.

Raises:

Type	Description
`InvalidParameterValueException`	Invalid number of classes requested.
`InvalidRasterBandException`	Input raster has more than one band.

7.12.2 Parameters

`FIRST_ORDER(RASTER, PARAMETERS, SCALING_FACTOR=1, SLOPE_TOLERANCE=0, SLOPE_GRADIENT_UNIT='RADIANS', SLOPE_DIRECTION_UNIT='RADIANS', METHOD='HORN')`

Calculate the first order surface attributes.

For compatibility for slope and aspect calculations with ArcGIS or QGIS, choose Method Horn (1981).

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Input raster.	required
`parameters`	`Sequence[Literal[G, A]]`	List of surface parameters to be calculated.	required
`scaling_factor`	`Optional[Number]`	Scaling factor to be applied to the raster data set. Default to 1.	`1`
`slope_tolerance`	`Optional[Number]`	Tolerance value for flat pixels. Default to 0.	`0`
`slope_gradient_unit`	`Literal[degrees, radians, rise]`	Unit of the slope gradient parameter. Default to radians.	`'radians'`
`slope_direction_unit`	`Literal[degrees, radians]`	Unit of the slope direction parameter. Default to radians.	`'radians'`
`method`	`Literal[Horn, Evans, Young, Zevenbergen]`	Method for calculating the coefficients. Default to the Horn (1981) method.	`'Horn'`

Returns:

Type	Description
`dict`	Selected surface attributes and respective updated metadata.

Raises:

Type	Description
`InvalidRasterBandException`	Raster has more than one band.
`NonSquarePixelSizeException`	Pixel dimensions do not have same length.
`InvalidParameterValueException`	Wrong input parameters provided.

`SECOND_ORDER_BASIC_SET(RASTER, PARAMETERS, SCALING_FACTOR=1, SLOPE_TOLERANCE=0, METHOD='YOUNG')`

Calculate the second order surface attributes.

References

Young, M., 1978: Terrain analysis program documentation. Report 5 on Grant DA-ERO-591-73-G0040, 'Statistical characterization of altitude matrices by computer'. Department of Geography, University of Durham, England: 27 pp.

Zevenbergen, L.W. and Thorne, C.R., 1987: Quantitative analysis of land surface topography, Earth Surface Processes and Landforms, 12: 47-56.

Wood, J., 1996: The Geomorphological Characterisation of Digital Elevation Models. Doctoral Thesis. Department of Geography, University of Leicester, England: 466 pp.

Parameters longc and crosc from are referenced by Zevenbergen & Thorne (1987) as profile and plan curvature. For compatibility with ArcGIS, choose Method Zevenbergen & Thorne (1987) and parameters longc and crosc.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Input raster.	required
`parameters`	`Sequence[Literal[planc, profc, profc_min, profc_max, longc, crosc, rot, K, genc, tangc]]`	List of surface parameters to be calculated.	required
`scaling_factor`	`Optional[Number]`	Scaling factor to be applied to the raster data set. Default to 1.	`1`
`slope_tolerance`	`Optional[Number]`	Tolerance value for flat pixels. Default to 0.	`0`
`method`	`Literal[Evans, Young, Zevenbergen]`	Method for calculating the coefficients. Default to the Young (1978) method.	`'Young'`

Returns:

Type	Description
`dict`	Selected surface attributes and respective updated metadata.

Raises:

Type	Description
`InvalidRasterBandException`	Raster has more than one band.
`NonSquarePixelSizeException`	Pixel dimensions do not have same length.
`InvalidParameterValueException`	Wrong input parameters provided.

7.12.3 Partial derivatives

7.12.4 Utilities

7.13 Filters

7.13.1 Focal

`FOCAL_FILTER(RASTER, METHOD='MEAN', SIZE=3, SHAPE='CIRCLE')`

Apply a basic focal filter to the input raster.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`method`	`Literal[mean, median]`	The method to use for filtering. Can be either "mean" or "median". Default to "mean".	`'mean'`
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`shape`	`Literal[square, circle]`	The shape of the filter window. Can be either "square" or "circle". Default to "circle".	`'circle'`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number. If the shape is not "square" or "circle".

`GAUSSIAN_FILTER(RASTER, SIGMA=1, TRUNCATE=4, SIZE=NONE)`

Apply a gaussian filter to the input raster.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`sigma`	`Number`	The standard deviation of the gaussian kernel.	`1`
`truncate`	`Number`	The truncation factor for the gaussian kernel based on the sigma value. Only if size is not given. Default to 4.0. E.g., for sigma = 1 and truncate = 4.0, the kernel size is 9x9.	`4`
`size`	`Optional[int]`	The size of the filter window. E.g., 3 means a 3x3 window. If size is not None, it overrides the dynamic size calculation based on sigma and truncate. Default to None.	`None`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number. If the resulting radius is smaller than 1.

`MEXICAN_HAT_FILTER(RASTER, SIGMA=1, TRUNCATE=4, SIZE=NONE, DIRECTION='CIRCULAR')`

Apply a mexican hat filter to the input raster.

Circular: Lowpass filter for smoothing. Rectangular: Highpass filter for edge detection. Results may need further normalization.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`sigma`	`Number`	The standard deviation.	`1`
`truncate`	`Number`	The truncation factor. E.g., for sigma = 1 and truncate = 4.0, the kernel size is 9x9. Default to 4.0.	`4`
`size`	`Optional[int]`	The size of the filter window. E.g., 3 means a 3x3 window. Default to None.	`None`
`direction`	`Literal[rectangular, circular]`	The direction of calculating the kernel values. Can be either "rectangular" or "circular". Default to "circular".	`'circular'`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number. If the resulting radius is smaller than 1.

7.13.2 Kernels

7.13.3 Speckle

`FROST_FILTER(RASTER, SIZE=3, DAMPING_FACTOR=1.0)`

Apply a Frost filter to the input raster.

Higher damping factor result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`damping_factor`	`Number`	Extent of exponential damping effect on filtering. Larger damping values preserve edges better but smooths less. Smaller values produce more smoothing. Default to 1.	`1.0`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`GAMMA_FILTER(RASTER, SIZE=3, N_LOOKS=1)`

Apply a Gamma filter to the input raster.

Higher number of looks result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`n_looks`	`int`	Number of looks to estimate the noise variation. Higher values result in higher smoothing. Low values may result in focal mean filtering. Default to 1.	`1`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`KUAN_FILTER(RASTER, SIZE=3, N_LOOKS=1)`

Apply a Kuan filter to the input raster.

Higher number of looks result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`n_looks`	`int`	Number of looks to estimate the noise variation. Higher values result in higher smoothing. Low values may result in focal mean filtering. Default to 1.	`1`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`LEE_ADDITIVE_MULTIPLICATIVE_NOISE_FILTER(RASTER, SIZE=3, ADD_NOISE_VAR=0.25, ADD_NOISE_MEAN=0, MULT_NOISE_MEAN=1)`

Apply a Lee filter considering both additive and multiplicative noise components in the input raster.

Lower noise values result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`add_noise_var`	`Number`	The additive noise variation. Default to 0.25.	`0.25`
`add_noise_mean`	`Number`	The additive noise mean. Default to 0.	`0`
`mult_noise_mean`	`Number`	The multiplative noise mean. Default to 1.	`1`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`LEE_ADDITIVE_NOISE_FILTER(RASTER, SIZE=3, ADD_NOISE_VAR=0.25)`

Apply a Lee filter considering additive noise components in the input raster.

Lower noise values result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`add_noise_var`	`Number`	The additive noise variation. Default to 0.25.	`0.25`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`LEE_ENHANCED_FILTER(RASTER, SIZE=3, N_LOOKS=1, DAMPING_FACTOR=1.0)`

Apply an enhanced Lee filter to the input raster.

Higher number of looks and damping factor result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`n_looks`	`int`	Number of looks to estimate the noise variation. Higher values result in higher smoothing. Low values may result in focal mean filtering. Default to 1.	`1`
`damping_factor`	`Number`	Extent of exponential damping effect on filtering. Larger damping values preserve edges better but smooths less. Smaller values produce more smoothing. Default to 1.	`1.0`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

`LEE_MULTIPLICATIVE_NOISE_FILTER(RASTER, SIZE=3, MULT_NOISE_MEAN=1, N_LOOKS=1)`

Apply a Lee filter considering multiplicative noise components in the input raster.

Higher number of looks result in better edge preservation.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	The input raster dataset.	required
`size`	`int`	The size of the filter window. E.g., 3 means a 3x3 window. Default to 3.	`3`
`mult_noise_mean`	`Number`	The multiplative noise mean. Default to 1.	`1`
`n_looks`	`int`	Number of looks to estimate the noise variation. Higher values result in higher smoothing. Default to 1.	`1`

Returns:

Type	Description
`tuple[ndarray, dict]`	The filtered raster array.

Raises:

Type	Description
`InvalidRasterBandException`	If the input raster has more than one band.
`InvalidParameterValueException`	If the filter size is smaller than 3. If the filter size is not an odd number.

7.13.4 Utilities

8. Training data tools

8.1 Class balancing

`balance_SMOTETomek(X, y, sampling_strategy='auto', random_state=None)`

Balances the classes of input dataset using SMOTETomek resampling method.

Parameters:

Name	Type	Description	Default
`X`	`Union[DataFrame, ndarray]`	The feature matrix (input data as a DataFrame).	required
`y`	`Union[Series, ndarray]`	The target labels corresponding to the feature matrix.	required
`sampling_strategy`	`Union[float, str, dict]`	Parameter controlling how to perform the resampling. If float, specifies the ratio of samples in minority class to samples of majority class, if str, specifies classes to be resampled ("minority", "not minority", "not majority", "all", "auto"), if dict, the keys should be targeted classes and values the desired number of samples for the class. Defaults to "auto", which will resample all classes except the majority class.	`'auto'`
`random_state`	`Optional[int]`	Parameter controlling randomization of the algorithm. Can be given a seed (number). Defaults to None, which randomizes the seed.	`None`

Returns:

Type	Description
`tuple[Union[DataFrame, ndarray], Union[Series, ndarray]]`	Resampled feature matrix and target labels.

Raises:

Type	Description
`NonMatchingParameterLengthsException`	If X and y have different length.

9. Transformations

9.1 Binarize

`binarize(raster, thresholds, bands=None, nodata=None)`

Binarize data based on a given threshold.

Replaces values less or equal threshold with 0. Replaces values greater than the threshold with 1.

Takes one nodata value which will be re-written after transformation.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands. The threshold can be set for each band individually.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`thresholds`	`Sequence[Number]`	Threshold values for transformation.	required
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.

9.2 Clip

`clip_transform(raster, limits, bands=None, nodata=None)`

Clips data based on specified upper and lower limits.

Takes one nodata value that will be ignored in calculations. Replaces values below the lower limit and above the upper limit with provided values, respecively. Works both one-sided and two-sided but raises error if no limits provided.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands. The limits can be set for each band individually.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`limits`	`Sequence[Tuple[Optional[Number], Optional[Number]]]`	Lower and upper limits (lower, upper) as real values.	required
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.
`InvalidParameterValueException`	The input does not match the requirements (values, order of values).

9.3 Linear

`min_max_scaling(raster, bands=None, new_range=[(0, 1)], nodata=None)`

Normalize data based on a specified new range.

Uses the provided new minimum and maximum to transform data into the new interval. Takes one nodata value that will be ignored in calculations.

If no band/column selection specified, all bands/columns will be used. The new_range can be set for each band individually. If a parameter contains only 1 entry, it will be applied for all bands.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`new_range`	`Sequence[Tuple[Number, Number]]`	The new interval data will be transformed into. First value corresponds to min, second to max.	`[(0, 1)]`
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.
`InvalidParameterValueException`	The input does not match the requirements (values, order of values).

`z_score_normalization(raster, bands=None, nodata=None)`

Normalize data based on mean and standard deviation.

Results will have a mean = 0 and standard deviation = 1. Takes one nodata value that will be ignored in calculations.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.

9.4 Logarithmic

```
log_transform(raster, bands=None, log_transform=['log2'], nodata=None)
```

Perform a logarithmic transformation on the provided data.

Takes one nodata value that will be ignored in calculations. Negative values will not be considered for transformation and replaced by the specific nodata value.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands. The log_transform can be set for each band individually.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`log_transform`	`Sequence[str]`	The base for logarithmic transformation. Valid values 'ln', 'log2' and 'log10'.	`['log2']`
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands
`InvalidParameterValueException`	The input does not match the requirements (values, order of values)

9.5 One-hot encoding

```
one_hot_encode(data, columns=None, drop_original_columns=True, drop_category=None, sparse_output=True, out_dtype=int, handle_unknown='infrequent_if_exist', min_frequency=None, max_categories=None)
```

Perform one-hot (or one-of-K or dummy) encoding on categorical data in a DataFrame or NumPy array.

This function converts categorical variables into a form that could be provided to machine learning algorithms for better prediction. For each unique category in the feature, a new binary column is created.

Continuous data should not be given to this function to avoid excessive amounts of binary features. If input is a DataFrame, continuous data can be excluded from encoding by specifying columns to encode.

The function allows control over aspects like handling unknown categories, controlling sparsity of the output, and setting data type of the encoded columns.

Parameters:

Name	Type	Description	Default
`data`	`Union[DataFrame, ndarray]`	Input data as a DataFrame or Numpy array. If a DataFrame is provided, the operation can be restricted to specified columns.	required
`columns`	`Optional[Sequence[str]]`	Specifies the columns to encode if 'data' is a DataFrame. If None, all columns are considered for encoding. Ignored if 'data' is a Numpy array. Defaults to None.	`None`
`drop_original_columns`	`bool`	If True and 'data' is a DataFrame, the original columns being encoded will be dropped from the output. Defaults to True.	`True`
`drop_category`	`Optional[Literal[first, if_binary]]`	Specifies a method to drop one of the categories to avoid multicollinearity. 'first' drops the first category, 'if_binary' drops one category only if the feature is binary. If None, no category is dropped. Defaults to None.	`None`
`sparse_output`	`bool`	Determines whether the output matrix is sparse or dense. Defaults to True (sparse).	`True`
`out_dtype`	`Union[type, dtype]`	Numeric data type of the output. Defaults to int.	`int`
`handle_unknown`	`Literal[error, ignore, infrequent_if_exist]`	Specifies how to handle unknown categories encountered during transform. 'error' raises an error, 'ignore' ignores unknown categories, and 'infrequent_if_exist' treats them as infrequent. Defaults to 'infrequent_if_exist'.	`'infrequent_if_exist'`
`min_frequency`	`Optional[Number]`	The minimum frequency (as a float or an int) needed to include a category in encoding. Optional parameter. Defaults to None.	`None`
`max_categories`	`Optional[int]`	The maximum number of categories to include in encoding. Optional parameter. Defaults to None.	`None`

Returns:

Type	Description
`Union[DataFrame, ndarray, csr_matrix]`	Encoded data as a DataFrame if input was a DataFrame, or as a Numpy array (dense or sparse) if input was a Numpy array.

Raises:

Type	Description
`EmptyDataFrameException`	If the input DataFrame is empty.
`InvalidDatasetException`	If the input Numpy array is empty.
`InvalidColumnException`	If any specified column to encode does not exist in the input DataFrame.

9.6 Sigmoid

```
sigmoid_transform(raster, bands=None, bounds=[(0, 1)], slope=[1], center=True, nodata=None)
```

Transform data into a sigmoid-shape based on a specified new range.

Uses the provided new minimum and maximum, shift and slope parameters to transform the data. Takes one nodata value that will be ignored in calculations.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands. The bounds and slope values can be set for each band individually.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`bounds`	`Sequence[Tuple[Number, Number]]`	Boundaries for the calculation of the sigmoid function (lower, upper).	`[(0, 1)]`
`slope`	`Sequence[Number]`	Value which modifies the slope of the resulting sigmoid-curve.	`[1]`
`center`	`bool`	Center array values around mean = 0 before sigmoid transformation.	`True`
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.
`InvalidParameterValueException`	The input does not match the requirements (values, order of values)

9.7 Winsorize

`winsorize(raster, percentiles, bands=None, inside=False, nodata=None)`

Winsorize data based on specified percentile values.

Takes one nodata value that will be ignored in calculations. Replaces values between [minimum, lower percentile] and [upper percentile, maximum] if provided. Works both one-sided and two-sided but raises error if no percentile values provided.

Percentiles are symmetrical, i.e. percentile_lower = 10 corresponds to the interval [min, 10%]. And percentile_upper = 10 corresponds to the intervall [90%, max]. I.e. percentile_lower = 0 refers to the minimum and percentile_upper = 0 to the data maximum.

Calculation of percentiles is ambiguous. Users can choose whether to use the value for replacement from inside or outside of the respective interval. Example: Given the np.array[5 10 12 15 20 24 27 30 35] and percentiles(10, 10), the calculated percentiles are (5, 35) for inside and (10, 30) for outside. This results in [5 10 12 15 20 24 27 30 35] and [10 10 12 15 20 24 27 30 30], respectively.

If no band/column selection specified, all bands/columns will be used. If a parameter contains only 1 entry, it will be applied for all bands. The percentiles can be set for each band individually, but inside parameter is same for all bands.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Data object to be transformed.	required
`bands`	`Optional[Sequence[int]]`	Selection of bands to be transformed.	`None`
`percentiles`	`Sequence[Tuple[Optional[Number], Optional[Number]]]`	Lower and upper percentile values (lower, upper) between [0, 100].	required
`inside`	`bool`	Whether to use the value for replacement from the left or right of the calculated percentile.	`False`
`nodata`	`Optional[Number]`	Nodata value to be considered.	`None`

Returns:

Name	Type	Description
`out_array`	`ndarray`	The transformed data.
`out_meta`	`dict`	Updated metadata.
`out_settings`	`dict`	Log of input settings and calculated statistics if available.

Raises:

Type	Description
`InvalidRasterBandException`	The input contains invalid band numbers.
`NonMatchingParameterLengthsException`	The input does not match the number of selected bands.
`InvalidParameterValueException`	The input does not match the requirements (values, order of values)

9.8 Coda

9.8.1 Additive logratio transform

`ALR_TRANSFORM(DF, COLUMN=NONE, KEEP_DENOMINATOR_COLUMN=FALSE)`

Perform an additive logratio transformation on the data.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of compositional data.	required
`column`	`Optional[str]`	The name of the column to be used as the denominator column.	`None`
`keep_denominator_column`	`bool`	Whether to include the denominator column in the result. If True, the returned dataframe retains its original shape.	`False`

Returns:

Type	Description
`DataFrame`	A new dataframe containing the ALR transformed data.

Raises:

Type	Description
`InvalidColumnException`	The input column isn't found in the dataframe.
`InvalidCompositionException`	Data is not normalized to the expected value.
`NumericValueSignException`	Data contains zeros or negative values.

`INVERSE_ALR(DF, DENOMINATOR_COLUMN, SCALE=1.0)`

Perform the inverse transformation for a set of ALR transformed data.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of ALR transformed compositional data.	required
`denominator_column`	`str`	The name of the denominator column.	required
`scale`	`Number`	The value to which each composition should be normalized. Eg., if the composition is expressed as percentages, scale=100.	`1.0`

Returns:

Type	Description
`DataFrame`	A dataframe containing the inverse transformed data.

Raises:

Type	Description
`NumericValueSignException`	The input scale value is zero or less.

9.8.2 Centered logratio transform

`CLR_TRANSFORM(DF)`

Perform a centered logratio transformation on the data.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of compositional data.	required

Returns:

Type	Description
`DataFrame`	A new dataframe containing the CLR transformed data.

Raises:

Type	Description
`InvalidCompositionException`	Data is not normalized to the expected value.
`NumericValueSignException`	Data contains zeros or negative values.

`INVERSE_CLR(DF, COLNAMES=NONE, SCALE=1.0)`

Perform the inverse transformation for a set of CLR transformed data.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of CLR transformed compositional data.	required
`colnames`	`Optional[Sequence[str]]`	List of column names to rename the columns to.	`None`
`scale`	`Number`	The value to which each composition should be normalized. Eg., if the composition is expressed as percentages, scale=100.	`1.0`

Returns:

Type	Description
`DataFrame`	A dataframe containing the inverse transformed data.

Raises:

Type	Description
`NumericValueSignException`	The input scale value is zero or less.

9.8.3 Isometric logratio transform

`SINGLE_ILR_TRANSFORM(DF, SUBCOMPOSITION_1, SUBCOMPOSITION_2)`

Perform a single isometric logratio transformation on the provided subcompositions.

Returns ILR balances. Column order matters.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of shape [N, D] of compositional data.	required
`subcomposition_1`	`Sequence[str]`	Names of the columns in the numerator part of the ratio.	required
`subcomposition_2`	`Sequence[str]`	Names of the columns in the denominator part of the ratio.	required

Returns:

Type	Description
`Series`	A series of length N containing the transforms.

Raises:

Type	Description
`InvalidColumnException`	One or more subcomposition columns are not found in the input dataframe.
`InvalidCompositionException`	Data is not normalized to the expected value or one or more columns are found in both subcompositions.
`InvalidParameterValueException`	At least one subcomposition provided was empty.
`NumericValueSignException`	Data contains zeros or negative values.

9.8.4 Pairwise logratio transform

`PAIRWISE_LOGRATIO(DF, NUMERATOR_COLUMN, DENOMINATOR_COLUMN)`

Perform a pairwise logratio transformation on the given columns.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	The dataframe containing the columns to use in the transformation.	required
`numerator_column`	`str`	The name of the column to use as the numerator column.	required
`denominator_column`	`str`	The name of the column to use as the denominator.	required

Returns:

Type	Description
`Series`	A series containing the transformed values.

Raises:

Type	Description
`InvalidColumnException`	One or both of the input columns are not found in the dataframe.
`InvalidParameterValueException`	The input columns contain at least one zero value.

`SINGLE_PAIRWISE_LOGRATIO(NUMERATOR, DENOMINATOR)`

Perform a pairwise logratio transformation on the given values.

Parameters:

Name	Type	Description	Default
`numerator`	`Number`	The numerator in the ratio.	required
`denominator`	`Number`	The denominator in the ratio.	required

Returns:

Type	Description
`float64`	The transformed value.

Raises:

Type	Description
`InvalidParameterValueException`	One or both input values are zero.

9.8.5 Pivot logratio transform

`PLR_TRANSFORM(DF)`

Perform a pivot logratio transformation on the dataframe, returning the full set of transforms.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of shape [N, D] of compositional data.	required

Returns:

Type	Description
`DataFrame`	A dataframe of shape [N, D-1] containing the set of PLR transformed data.

Raises:

Type	Description
`InvalidColumnException`	The data contains one or more zeros.
`InvalidCompositionException`	Data is not normalized to the expected value.
`NumericValueSignException`	Data contains zeros or negative values.

`SINGLE_PLR_TRANSFORM(DF, COLUMN)`

Perform a pivot logratio transformation on the selected column.

Pivot logratio is a special case of ILR, where the numerator in the ratio is always a single part and the denominator all of the parts to the right in the ordered list of parts.

Column order matters.

Parameters:

Name	Type	Description	Default
`df`	`DataFrame`	A dataframe of shape [N, D] of compositional data.	required
`column`	`str`	The name of the numerator column to use for the transformation.	required

Returns:

Type	Description
`Series`	A series of length N containing the transforms.

Raises:

Type	Description
`InvalidColumnException`	The input column isn't found in the dataframe, or there are no columns to the right of the given column.
`InvalidCompositionException`	Data is not normalized to the expected value.
`NumericValueSignException`	Data contains zeros or negative values.

10. Utilities

10.1 File I/O utilities

`get_output_paths_from_common_name(outputs, directory, common_name, extension, first_num=1)`

Get output paths for cases where outputs should be just numbered.

Combines directory, given common file name, number and extension. Outputs are used to get the number used as suffix. Include dot in the extension, for example '.tif'.

This tool is designed mainly for convenience in CLI functions.

Parameters:

Name	Type	Description	Default
`input_paths`		Outputs. Used just to iterate and get numbers for suffixes.	required
`directory`	`Path`	Path of the output directory.	required
`common_name`	`str`	Common name used as the basis of each output file name. A number is appended to this.	required
`extension`	`str`	The extension used for the output path, for example ".tif".	required
`first_num`	`int`	The first number used as a suffix.	`1`

Returns:

Type	Description
`Sequence[Path]`	List of output paths.

`get_output_paths_from_inputs(input_paths, directory, suffix, extension)`

Get output paths using input paths to extract file name bases.

Combines directory, file name extracted from input path, suffix and extension. Include dot in the extension, for example '.tif'.

This tool is designed mainly for convenience in CLI functions.

Parameters:

Name	Type	Description	Default
`input_paths`	`Sequence[Path]`	Input paths.	required
`directory`	`Path`	Path of the output directory.	required
`suffix`	`str`	Common suffix added to the end of each output file name, for example "nodata_unified".	required
`extension`	`str`	The extension used for the output path, for example ".tif".	required

Returns:

Type	Description
`Sequence[Path]`	List of output paths.

`get_output_paths_from_names(file_names, directory, suffix, extension)`

Get output paths directly from given file names.

Combines directory, file name, suffix and extension. Include dot in the extension, for example '.tif'.

This tool is designed mainly for convenience in CLI functions.

Parameters:

Name	Type	Description	Default
`input_paths`		Raw file names.	required
`directory`	`Path`	Path of the output directory.	required
`suffix`	`str`	Common suffix added to the end of each output file name, for example "nodata_unified".	required
`extension`	`str`	The extension used for the output path, for example ".tif".	required

Returns:

Type	Description
`Sequence[Path]`	List of output paths.

`read_and_stack_rasters(raster_files, nodata_handling='convert_to_nan')`

Read multiple raster files and stack all their bands into a single 3D array.

Checks that all rasters have the same grid properties. If there are any differences, exception is raised.

Parameters:

Name	Type	Description	Default
`raster_files`	`Sequence[Path]`	List of paths to raster files.	required
`nodata_handling`	`Literal[convert_to_nan, unify, raise_exception, none]`	How to handle raster nodata. convert_to_nan to changes all nodata to np.nan, unify changes all rasters to use -9999 as their nodata value, raise_exception raises an exception if all rasters do not have the same nodata value, and none does not do anything for nodata.	`'convert_to_nan'`

Returns:

Type	Description
`ndarray`	3D array with shape (total bands, height, width).
`Sequence[Profile]`	List of raster profiles.

Raises:

Type	Description
`NonMatchingRasterMetadataException`	If input rasters do not have same grid properties or nodata_handling is set to raise exception and mismatching nodata is encountered.

`read_file(file_path)`

Read an input file trying different readers.

First tries to read to a rasterio DatasetReader, then to a GeoDataFrame, then to a DataFrame. If none of the readers succeed, raises an exception.

Parameters:

Name	Type	Description	Default
`file_path`	`Path`	Input file path.	required

Returns:

Type	Description
`Union[DatasetReader, GeoDataFrame, DataFrame]`	The input file data in the opened format.

Raises:

Type	Description
`FileReadError`	None of the readers succeeded to read the input file.

`read_raster(file_path)`

Read a raster file to a rasterio DatasetReader.

Parameters:

Name	Type	Description	Default
`file_path`	`Path`	Input file path.	required

Returns:

Type	Description
`DatasetReader`	File data as a Rasterio DatasetReader.

Raises:

Type	Description
`FileReadError`	Rasterio failed to open the input file.

`read_tabular(file_path)`

Read tabular data to a DataFrame.

Parameters:

Name	Type	Description	Default
`file_path`	`Path`	Input file path.	required

Returns:

Type	Description
`DataFrame`	File data as a DataFrame.

Raises:

Type	Description
`FileReadError`	Pandas failed to open the input file.

`read_vector(file_path)`

Read a vector file to a GeoDataFrame.

Parameters:

Name	Type	Description	Default
`file_path`	`Path`	Input file path.	required

Returns:

Type	Description
`GeoDataFrame`	File data as a GeoDataFrame.

Raises:

Type	Description
`FileReadError`	Geopandas failed to read the input file.

10.2 Nodata utilities

`convert_raster_nodata(input_raster, old_nodata=None, new_nodata=-9999)`

Convert existing nodata values with a new nodata value for a raster.

Parameters:

Name	Type	Description	Default
`input_raster`	`DatasetReader`	Input raster dataset.	required
`new_nodata`	`Number`	New nodata value that will be used to replace existing nodata for all bands. Defaults to -9999.	`-9999`

Returns:

Type	Description
`Tuple[ndarray, dict]`	The input raster data and metadata updated with the new nodata.

Raises:

Type	Description
`InvalidParameterValueException`	Nodata is not defined in raster metadata and old_nodata was not specified.

`handle_nodata_as_nan(func)`

Replace nodata_values with np.nan for function execution and reverses the replacement afterwards.

`nan_to_nodata(data, nodata_value)`

Convert np.nan values to specified nodata_value.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	Input data as a numpy array.	required
`nodata_value`	`Number`	Value that np.nan is converted to.	required

Returns:

Type	Description
`ndarray`	Input array where np.nan has been converted to specified nodata.

`nodata_to_nan(data, nodata_value)`

Convert specified nodata_value to np.nan.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	Input data as a numpy array.	required
`nodata_value`	`Number`	Value that is converted to np.nan.	required

Returns:

Type	Description
`ndarray`	Input array where specified nodata has been converted to np.nan.

`set_raster_nodata(raster_meta, new_nodata)`

Set new nodata value for raster metadata or profile.

Note that this function does not convert any data values, it only changes the metadata. The inteded use case for this tool is fixing metadata with invalid nodata value.

Parameters:

Name	Type	Description	Default
`raster_meta`	`Union[Dict, Profile]`	Raster metadata or profile to be updated.	required
`nodata_value`		New nodata value.	required

Returns:

Type	Description
`Union[Dict, Profile]`	Raster metadata / profile with updated nodata value.

`unify_raster_nodata(input_rasters, new_nodata=-9999)`

Unifies nodata for the input rasters.

All old nodata values in the input rasters are converted to the new nodata value. Raster metadatas is also updated with the new nodata value.

Parameters:

Name	Type	Description	Default
`input_rasters`	`Sequence[DatasetReader]`		required
`new_nodata`	`Number`	New nodata value that will be used to replace existing nodata for all bands in all input rasters. Defaults to -9999.	`-9999`

Returns:

Type	Description
`Sequence[Tuple[ndarray, dict]]`	Output raster list. List elements are tuples where first element is raster data and second element is raster metadata.

Raises:

Type	Description
`InvalidParameterValueException`	Input raster list contains only one raster.

10.3 Raster data utilities

`combine_raster_bands(input_rasters)`

Combine multiple rasters into one multiband raster.

The input rasters can be either singleband or multiband. All bands are stacked in the order they are extracted from the input raster list.

All input rasters must have matching spatial metadata (extent, pixel size, CRS).

Parameters:

Name	Type	Description	Default
`input_rasters`	`Sequence[DatasetReader]`	List of rasters to combine.	required

Returns:

Type	Description
`ndarray`	The combined raster data.
`Profile`	The updated raster profile.

Raises:

Type	Description
`InvalidParameterValueException`	Input rasters contains only one raster.
`NonMatchingRasterMetadataException`	Input rasters have mismatching spatial metadata.

`profile_from_extent_and_pixel_size(extent, pixel_size, round_strategy='up')`

Create a raster profile from given extent and pixel size.

If extent and pixel size do not match exactly, i.e. raster width and height calcalated from bounds and pixel size are not integers, rounding for the width and height is performed.

Parameters:

Name	Type	Description	Default
`extent`	`Tuple[Number, Number, Number, Number]`	Raster extent in the form (coord_west, coord_east, coord_south, coord_north).	required
`pixel_size`	`Union[Number, Tuple[Number, Number]]`	Desired pixel size. If two values are provided, first is used for x and second for y. If one value is provided, the value is used for both directions.	required
`round_strategy`	`Literal[nearest, up, down]`	The rounding strategy if extent and pixel size do not match exactly. Defaults to "up".	`'up'`

Returns:

Type	Description
`Profile`	Rasterio profile.

`split_raster_bands(raster)`

Split multiband raster into singleband rasters.

Parameters:

Name	Type	Description	Default
`raster`	`DatasetReader`	Input multiband raster.	required

Returns:

Type	Description
`Sequence[Tuple[ndarray, Profile]]`	Output singleband raster list. List elements are tuples where first element is raster data (2D) and second element is raster profile.

Raises:

Type	Description
`InvalidParameterValueException`	Input raster contains only one band.

`stack_raster_arrays(arrays)`

Stack 2D and 3D NumPy arrays (each representing a raster with one or multiple bands) along the bands axis.

Parameters:

Name	Type	Description	Default
`arrays`	`Sequence[ndarray]`	List of 2D and 3D NumPy arrays. Each 2D array should have shape (height, width). and 3D array shape (bands, height, width).	required

Returns:

Type	Description
`ndarray`	A single 3D NumPy array where the first dimension size equals the total number of bands.

Raises:

Type	Description
`InvalidDataShapeException`	Input raster arrays have mismatching shapes or all input rasters are not 2D or 3D.

11. Vector processing

11.1 Calculate geometry

`calculate_geometry(geodataframe)`

Calculate the length or area of the given geometries.

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	Geometries to be calculated.	required

Returns:

Name	Type	Description
`calculated_gdf`	`GeoDataFrame`	Geometries and calculated values.

11.2 Cell-Based Association

`cell_based_association(cell_size, geodata, output_path, column=None, subset_target_attribute_values=None, add_name=None, add_buffer=None)`

Creation of CBA matrix.

Initializes a CBA matrix from a vector file. The mesh is calculated according to the geometries contained in this file and the size of cells. Allows to add multiple vector data to the matrix, based on targeted shapes and/or attributes.

Parameters:

Name	**Type**	**Description**	**Default**
`cell_size`	`int`	Size of the cells.	required
`geodata`	`List[GeoDataFrame]`	GeoDataFrame to create the CBA matrix. Additional GeoDataFrame(s) can be imputed to add to the CBA matrix.	required
`output_path`	`str`	Name of the saved .tif file.	required
`column`	`Optional[List[str]]`	Name of the column of interest. If no attribute is specified, then an artificial attribute is created representing the presence or absence of the geometries of this file for each cell of the CBA grid. A categorical attribute will generate as many columns (binary) in the CBA matrix than values considered of interest (dummification). See parameter . Additional column(s) can be imputed for each added GeoDataFrame(s).	`None`
`subset_target_attribute_values`	`Optional[List[Union[None, list, str]]]`	List of values of interest of the target attribute, in case a categorical target attribute has been specified. Allows to filter a subset of relevant values. Additional values can be imputed for each added GeoDataFrame(s).	`None`
`add_name`	`Optional[List[Union[str, None]]]`	Name of the column(s) to add to the matrix.	`None`
`add_buffer`	`Optional[List[Union[Number, bool]]]`	Allow the use of a buffer around shapes before the intersection with CBA cells for the added GeoDataFrame(s). Minimize border effects or allow increasing positive samples (i.e. cells with mineralization). The size of the buffer is computed using the CRS (if projected CRS in meters: value in meters).	`None`

Returns:

Type	Description
`GeoDataFrame`	CBA matrix is created.

11.3 Distance computation

`distance_computation(geodataframe, raster_profile)`

Calculate distance from raster cell to nearest geometry.

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	The GeoDataFrame with geometries to determine distance to.	required
`raster_profile`	`Union[Profile, dict]`	The raster profile of the raster in which the distances to the nearest geometry are determined.	required

Returns:

Type	Description
`ndarray`	A 2D numpy array with the distances computed.

Raises:

Type	Description
`NonMatchingCrsException`	The input raster profile and geodataframe have mismatching CRS.
`EmptyDataFrameException`	The input geodataframe is empty.

11.4 Extract shared lines

`extract_shared_lines(polygons)`

Extract shared lines/borders/edges between polygons.

Parameters:

Name	Type	Description	Default
`polygons`	`GeoDataFrame`	The geodataframe that contains the polygon geometries to be examined for shared lines.	required

Returns:

Type	Description
`GeoDataFrame`	Geodataframe containing the shared lines that were found between the polygons.

11.5 IDW

`idw(geodataframe, target_column, raster_profile, power=2)`

Calculate inverse distance weighted (IDW) interpolation.

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	The vector dataframe to be interpolated.	required
`target_column`	`str`	The column name with values for each geometry.	required
`raster_profile`	`Union[Profile, dict]`	The raster profile used for output grid properties. Needs to include at least crs, transform, width and height.	required
`power`	`Number`	The value for determining the rate at which the weights decrease. As power increases, the weights for distant points decrease rapidly. Defaults to 2.	`2`

Returns:

Type	Description
`ndarray`	Numpy array containing the interpolated values.

Raises:

Type	Description
`EmptyDataFrameException`	The input GeoDataFrame is empty.
`InvalidParameterValueException`	Invalid resolution or target_column.
`NonMatchingCrsException`	The input GeoDataFrame and raster profile have mismatching CRS.

11.6 Kriging interpolation

```
kriging(geodataframe, target_column, raster_profile, variogram_model='linear', coordinates_type='geographic', method='ordinary')
```

Perform Kriging interpolation on the input data.

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	GeoDataFrame containing the input data.	required
`target_column`	`str`	The column name with values for each geometry.	required
`raster_profile`	`Union[Profile, dict]`	The raster profile used for output grid properties. Needs to include at least crs, transform, width and height.	required
`variogram_model`	`Literal[linear, power, gaussian, spherical, exponential]`	Variogram model to be used. Either 'linear', 'power', 'gaussian', 'spherical' or 'exponential'. Defaults to 'linear'.	`'linear'`
`coordinates_type`	`Literal[euclidean, geographic]`	Determines are coordinates on a plane ('euclidean') or a sphere ('geographic'). Used only in ordinary kriging. Defaults to 'geographic'.	`'geographic'`
`method`	`Literal[ordinary, universal]`	Ordinary or universal kriging. Defaults to 'ordinary'.	`'ordinary'`

Returns:

Type	Description
`ndarray`	Numpy array containing the interpolated values.

Raises:

Type	Description
`EmptyDataFrameException`	The input GeoDataFrame is empty.
`InvalidParameterValueException`	Target column name is invalid or resolution is not greater than zero.
`NonMatchingCrsException`	The input GeoDataFrame and raster profile have mismatching CRS.

11.7 Rasterize vector

```
rasterize_vector(geodataframe, raster_profile, value_column=None, default_value=1.0, fill_value=0.0, buffer_value=None, merge_strategy='replace')
```

Transform vector data into raster data.

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	The vector dataframe to be rasterized.	required
`raster_profile`	`Union[Profile, dict]`	The raster profile used for output grid properties. Needs to include at least CRS, transform, width and height.	required
`value_column`	`Optional[str]`	The column name with values for each geometry. If None, then default_value is used for all geometries.	`None`
`default_value`	`float`	Default value burned into raster cells based on geometries.	`1.0`
`fill_value`	`float`	Value used outside the burned/rasterized geometry cells.	`0.0`
`buffer_value`	`Optional[float]`	For adding a buffer around passed geometries before rasterization.	`None`
`merge_strategy`	`Literal[replace, add]`	How to handle overlapping geometries. "add" causes overlapping geometries to add together the values while "replace" does not. Adding them together is the basis for density computations where the density can be calculated by using a default value of 1.0 and the sum in each cell is the count of intersecting geometries.	`'replace'`

Returns:

Type	Description
`ndarray`	Rasterized vector data.

Raises:

Type	Description
`EmptyDataFrameException`	The geodataframe does not contain geometries.
`InvalidColumnException`	Given value_column is not in the input geodataframe.
`NonMatchingCrsException`	The input GeoDataFrame and raster profile have mismatching CRS.
`NumericValueSignException`	Input resolution value is zero or negative, or input buffer_value is negative.

11.8 Reproject vector

`reproject_vector(geodataframe, target_crs)`

Reprojects vector data to match given coordinate reference system (EPSG).

Parameters:

Name	Type	Description	Default
`geodataframe`	`GeoDataFrame`	The vector dataframe to be reprojected.	required
`target_crs`	`int`	Target CRS as an EPSG code.	required

Returns:

Type	Description
`GeoDataFrame`	Reprojected vector data.

11.9 Vector density

```
vector_density(geodataframe, raster_profile, buffer_value=None, statistic='density')
```

Compute density of geometries within raster.

Parameters:

Name	**Type**	**Description**	**Default**
`geodataframe`	`GeoDataFrame`	The dataframe with vectors of which density is computed.	required
`base_raster_profile`		Base raster profile to be used for determining the grid on which vectors are burned in. If None, the geometries and provided resolution value are used to compute grid.	required
`buffer_value`	`Optional[float]`	For adding a buffer around passed geometries before computing density.	`None`
`statistic`	`Literal[density, count]`	The statistic to use in density computation. Defaults to "density".	`'density'`

Returns:

Type	**Description**
`ndarray`	Computed density of vector data.