Universidad de Oviedo

EPM ESCUELA POLITÉCNICA DE MIERES UNIVERSIDAD DE OVIEDO

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN
GEOTECNOLOGÍA Y DESARROLLO DE PROYECTOS SIG

Linea de Nieve:
Automated Snow Classification from Sentinel-2 Data Using Python
~
Clasificación Automatizada de Nieve a partir de Datos de Sentinel-2 Utilizando Python

Autor: Jakob POPPELLER, BSc

Tutor: D. CABO GÓMEZ, Carlos

Febrero, 2025

Abstract

Abstract in Spanish

Los métodos actuales para crear reportes diarios de aludes con el objetivo de informar a los montañeros y a los habitantes afectados por dicho peligro, se basan en diversos datos. Éstos datos incluyen informaciónes de las estaciones meteorológicas, observaciones del paisaje y simulaciones de la cobertura de nieve. La integración de un análisis basado en imágines satélites ofrece un gran potencial para mejorar la evaluación de riesgos de aludes, especialmente al principio y al final de la temporada cuando la capa de nieve está fragmentada.[1] Éstas herramientas son esenciales para identificar las zonas cubiertas de nieve y las zonas sin nieve que pueden ser una parte fundamental en la evaluación de los riesgos de aludes.
Éste trabajo fin de master se centra en el desarrollo de un sistema para procesar mapas de la capa de nieve actual y generar estadísticas detalladas sobre la distribución de la nieve utilizando datos de Sentinel-2. Éstas estadísticas tienen en cuenta las variaciones de altitud sobre el nivel del mar y orientación. Reconociendo que la alturas de las líneas de nieve son sustancialmente más altas en laderas soleadas y orientadas al sur que en laderas sombreadas y orientadas al norte. En este documento la línea de nieve se define como la línea de altitud entre nieve y ausencia de nieve.
El flujo de trabajo permite además generar estadísticas para cualquier zona de interés, garantizando un análisis preciso y localizado. El algoritmo utilizado para clasificar la nieve es una versión simplificada del algoritmo *Thiea Snow Collection*[2]. Los resultados de las clasificaciones son prometedores. En una verificación groundtruth se comparan determinados puntos del análisis con los puntos correspondientes de las imágenes de la webcam. Se alcanzó una tasa de error inferior al 2 %. Éstos errores se producen principalmente en zonas afectadas por nubes y sombras de nubes. Ya existen algoritmos de detección de nubes para las imágenes Sentinel 2 que pueden solucionar éste problema y que podrían aplicarse en futuros desarrollos.

Palabras clave: mapa de nieve, Sentinel 2, elevación, orientación

Abstract in English

Current methods to create daily avalanche bulletins, to inform mountaineers and inhabitants affected by avalnce danger, require many sources of inputs, including weather station data, field observations, and snowpack simulations. The integration of satellite-based analysis offers significant potential to enhance the monitoring process, particularly during early and late seasons when snow cover is fragmented.[1] These tools are essential for identifying snow-covered and snow-free areas, which can play a critical role in assessing avalanche risks.
This thesis focuses on the development of a processing pipeline maps of current snow cover and generate detailed statistics on snow distribution using Sentinel-2 data. These statistics take into account the variations in elevation and aspect because there are variations in the elevation line between the presence and absence of snow, further referenced as the snowline, on sunny, south-facing slopes compared to shaded, north-facing ones. This workflow generates statistics for any requested subregions, ensuring a precise and localized analysis.
The algorithm used to classify snow is a simplified version of the *Thiea Snow Collection* algorithm[2]. The results of the classifications are promising. In a groundtruth check where certain points in the analysis are compared to the corresponding points in webcam images. A misclassification rate of under 2%. These misclassifications are mainly in areas affected

by clouds and cloud shadows. To address this, there are already cloud detection algorithms developed for Sentinel 2 imagery that could be implemented for future development.

Key words: snow map, Sentinel 2, elevation, orientation.

Foreword and acknowledgements

I was introduced to the interesting and widespread topic of Remotesensing and GIS during my Esasmus stay in Oviedo. Thanks to the complex and intertwined Master in Innsbruck of civil- and environmental engineering, I managed that the university of Innsbruck agreed to a learning agreement that allowed me to take nearly all of the classes of this master in my year at the EPM in Mieres. Thanks that this worked out.
This was really great because I was able to make good connections with my classmates and was completely integrated into the study program of the EPM in Mieres.
At the end of my year in Spain, I consulted the university if I could finish this master and get an degree, given that I have taken already most of the courses. The first response was, that it is not possible, because I am enrolled in Innsbruck. But with the untiring help of Candela we found a way to append all my credit points to a regular study course.
At this point there was still the internship and the thesis missing.
I managed that I could make my internship in Innsbruck at the Avalanche Warning Service. Thanks the Avalanche Warning Service Team of Tirol, that I could do this internship and get some unique insights into the workflow of this fascinating job. It was an unforgettable year. Also special thanks to Patrick for writing and signing all the documents and evaluations for the recognition of the internship in university in Spain.
Toward the end of this internship, I started writing and coding on this thesis. Thanks to Azad for helping me start the project. He helped me a lot with his outstanding coding skills. Especially for the part of automatically downloading the requested files from the Copernicus homepage.
Thanks to Carlos for tutoring my work and making time for my correction on this very tight time schedule.
Then I want to thank my friends and family for supporting me in my projects.
Very special thanks to Christine for editing some parts of the work to improve my English and to support me in the last weeks, when the writing got more intense. I am very happy to have you in my life.

Declaration of Originality

I, Jakob Poppeller declare that i am the author of the Master's Thesis called "Linea de Nieve: Automated Snow Classification from Sentinel-2 Data Using Python". This work is original and all sources have been cited according to the academic standards.
This work has also not been submitted previously in any form for the attainment of any other academic degree.
Hall in Tirol, February 2025

Jakob Poppeller

Contents

Abstract 2
Abstract in Spanish . . . 2
Abstract in English . . . 2

Acknowledgements 4

Declaration of Originality 5

Table of contents 5

List of figures 8

List of tables 9

1 Introduction 10
1.1 State of the art . . . 10

2 Objectives 14

3 Methods 15
3.1 Detection Method used in this Thesis . . . 15
3.1.1 Inputs . . . 16
3.1.2 Snowclasification . . . 16
3.2 Version Control and Documentation . . . 17
3.3 Workflow and Configuration . . . 18
3.3.1 Configuration . . . 21
3.3.2 Downloads and Preprocessing . . . 22
3.3.3 Processing . . . 23
3.3.4 Postprocessing . . . 25
3.3.5 Statistics . . . 26
3.4 Validation using Webcam Images . . . 27

4 Results and Discussion 31
4.1 Results of the Groundtruth analysis . . . 31
4.2 Discussion of the Output . . . 32
4.3 Limitations and Challenges . . . 34

5 Conclusion 37
5.1 Summary of the developed Code . . . 37
5.2 Summary of the experimental results . . . 37
5.3 Future Work . . . 38

6 Conclusiónes (Síntesis en Español) 40

Bibliografía 41

Appendices **43**
6.0.1 Code sample and explanation of the `_501_get_microregion_stats.py` script 43

List of Figures

1 Example of the SNOWGRID Simulation from the Website of the avalanche warning service.[3] . . . 12
2 Comparison to Webcam image of the same day. [4] . . . 12

3 Illustration of the refectance spectrum of snow and the used Bands for the NDSI[5] . . . 17
4 Overview of the scripts in the Gitlab reopsitory. . . . 18
5 https://gitlab.com/Jakob_Poppeller/linea_de_nieve . . . 19
6 Flowchart of the algorithms Workflow. . . . 20
7 Overview of the structure from Downloads and Preprocessing . . . 22
8 Overview of the structure from Downloads and Preprocessing . . . 24
9 Example of a pixel value in the merged file . . . 26
10 Overview or the Webcam positions in Austria . . . 28
11 Example of the Controlpoints and Groundtruth Analysis San Isidro - León . 29
12 Example of the Controlpoints and Groundtruth Analysis Kitzbühler Horn - Tirol . . . 30

13 Overview of the analyzed area . . . 32
14 Example of the Results Statistics File with one example region highlighted . 32
15 Overview of the aspects and analysis of the Nordkette (Region AT-07-04-01_Westliches Karwendel) . . . 34
16 Diagram of the distribution of the Snow in the study area . . . 35
17 Webcam footage of the study area with avalanche pathways . . . 35
18 Overview of the wrongly classified points due to Cloud-shadow . . . 36

19 Example for the classification of small areas of snow.
On the left: the analysis (snow - turquoise; no snow - magenta)
On the right: corresponding webcam image.[4] . . . 38
20 Sketch of a graphical representation of the snowline . . . 39

List of Tables

4.1 Results of ground truth analysis. 31

1. Introduction

Numerous applications across various professional fields, particularly in avalanche prediction, monitor actual snow cover.
The data used to generate daily avalanche bulletins are from multiple sources, including weather stations distributed throughout the warning area, field observations from professionals such as mountain guides, hut staff, ski resort workers, and members of avalanche commissions. Additional information is derived from snowpack simulations and on-site observations, and even webcam data can be helpful.
The integration of satellite-based analysis offers significant potential for improving snow cover monitoring. These tools could be particularly valuable during the early and late seasons, when snow cover is fragmented, making the identification of snow-covered and snow-free areas essential.
If early season snowfall is followed by an extended period of stable, high-pressure weather or precipitation-free weather, there is the possibility that a deep persistent weak layer develops. Subsequent snowfall can be deposited in that weak layer, resulting in a dangerous avalanche situation.[1] Therefore, it is important to know where there was some snow left before the next snowfall.
In such cases, after a new snowfall, field tests can be performed to assess the stability of the snowpack in areas previously covered with snow.
The snowline information is also valuable in the spring, when there is no snow below a certain elevation threshold, as it can be used to establish a lower boundary for the actual warning area.
Mountaineers rely on accurate information about the snowline, particularly during spring conditions, to plan their activities in detail and determine whether specific areas remain snow covered. This information is valuable for climbers to assess whether routes are free of snow, as well as for backcountry splitboarders and skiers to determine where to start their approach and how far they need to go by foot or by bike to reach the snow. There is a high demand for this data, and it would improve the practicality of planning tools. Therefore, data must be preprocessed and provided on an easily accessible platform, such as a website, since conducting individual analyses is too resource-intensive.

1.1.- State of the art methods of Snow Monitoring

Many techniques and approaches have been developed over the years to represent snow cover, ranging from traditional manual measurements to advanced simulations and satellite-based remote sensing. Each method has its strengths, limitations, and areas of application.

The oldest and still widely practiced method involves measuring snow depth at various weather stations in different locations, providing point-based samples of the current snow conditions.
This can be done manually, by measuring the accumulation of snow each day at the same time. This is still be done [6], but it is very time consuming and can only be done in places where there are people around all year.
There are also automatic stations to determine the snow depth. In this setup an ultrasonic or laser sensor mounted above the snow cover measures the distance to the snow surface[7]. These automatic measurement apparatuses can be installed in remote and high elevation areas. For example there is over a 160 stations monitoring the snow height in the state of Tirol.[1] [8]. Also Switzerland has around 130 snowmeasuring stations. This creates a dense network of point samples in the alpine regions, especially in the higher elevated regions that are harder to access.[7]
While this approach delivers even the depth of the snow at specific locations, it is limited by its inability to represent spatial variability comprehensively. Despite this limitation, snow depth measurements remain a valuable and reliable source of data, offering unmatched precision for localized observations.

Snow cover can also be modeled through simulations, such as the SNOWGRID model [9], developed by GeoSphere Austria. This model integrates data from weather simulations and weather station observations. With a resolution of up to 100 meters and simulations performed in 15-minute intervals, SNOWGRID incorporates the physical properties of snow into its calculations [10], providing a powerful tool to analyze the snow cover and depth over an extended area. This simulation is used to generate the snow maps of the avalanche warning service. Figure 1 shows the SNOWGRID simulation combined with the point measurements of the weather stations displayed in the circles. The figure shows the analysis from January 17, 2025, and can be compared with Figure 2, the webcam image of the same day. The view of the webcam is also pictured in the image of the simulation. It can be observed, that the simulation often still shows thin snow cover in the border areas of the snow, although the regions are already free of snow.

Several snow classification products are available for remote sensing applications. The Fractional Snow Cover (FSC) [11] product, developed within the Copernicus High-Resolution Snow and Ice Monitoring framework, provides a spatial resolution of 20x20 m. It employs

[1]The data can be found on avalanche.report/weather/measurements

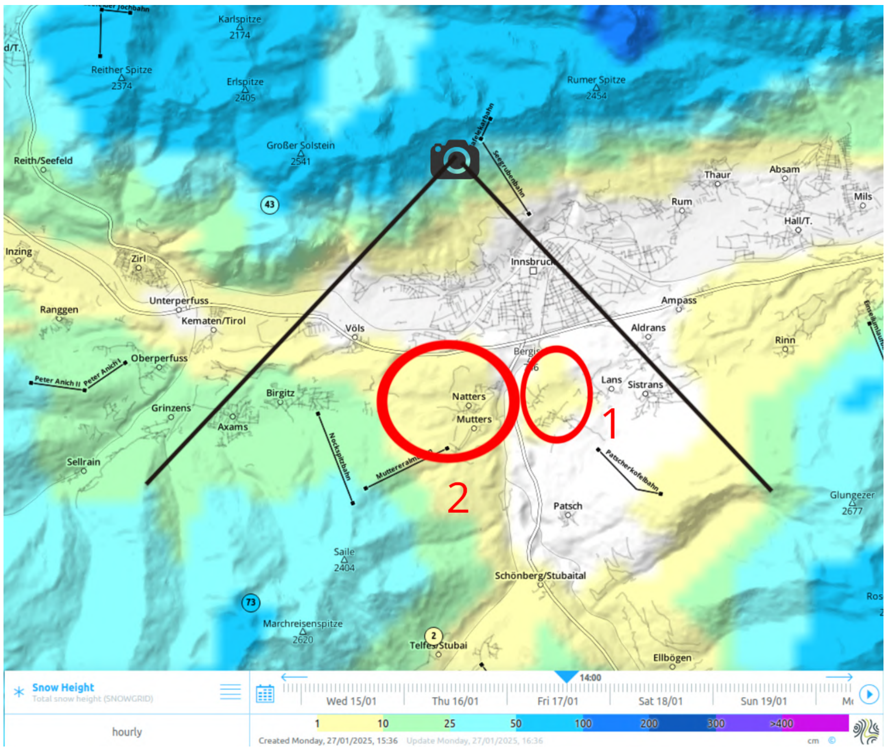


Figure 1: Example of the SNOWGRID Simulation from the Website of the avalanche warning service.[3]

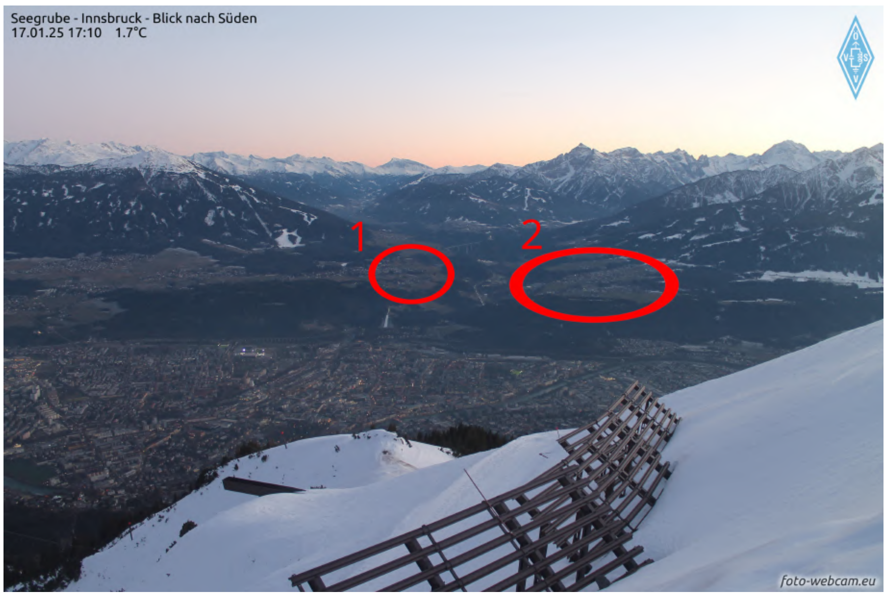


Figure 2: Comparison to Webcam image of the same day. [4]

the Let It Snow (LIS) algorithm, developed by Theia [2], to determine snow presence and compute fractional snow cover using optical bands, relying primarily on the Normalized Difference Snow Index (NDSI)[12] for classification. However, due to the reliance on optical data, snow under cloud cover cannot be detected.

Additionally, the Gap-filled Fractional Snow Cover product [13], also developed by Copernicus, addresses this limitation by combining FSC with Synthetic Aperture Radar (SAR) data. This product achieves a spatial resolution of 60x60 m and effectively fills gaps caused by cloud cover, providing more comprehensive snow cover mapping. By compromising on the spatial resolution of the image due to the resolution of the radar sensor.

2. Objectives

The objective of this thesis is to develop a method for generating maps of the current snow cover. In addition, it aims to produce statistics on the snow distribution while taking into account the aspect and elevation.

The method must support the subdivision of the study area into small individual regions. The subdivision is crucial because snow conditions can vary significantly across areas.

It is also a priority to use freely available data sources with high spatial and temporal resolution and use open-source software for the analysis.

Moreover, the results should be designed for future integration into the website of the avalanche warning service, www.lawinen.report. Therefor the code should run autonomously without further user interaction, so that it can be implemented into automatic workflows of daily updates.

The workflow should be optimized to update the data incrementally, avoiding the need to regenerate all outputs from scratch during each execution. This design will reduce processing time and data usage.

The output format must be designed for compatibility with other software, enabling full automation of subsequent processes. This ensures that the generated data can be integrated into broader workflows for snowline analysis and integrated on the website.

3. Methods

The code developed for this thesis can process Sentinel 2 imagery and automates the classification of snow cover. After setting up the configurations, it is possible to run the code by executing one command to meet the objectives for the implementation into automatized workflows that update daily. If there is new data available it automatically updates the maps and statistical output.
Despite the availability of existing programs for snow classification[2][11][13], this code has been freshly developed for two main reasons.
First, it ensures a comprehensive understanding of the processing steps involved. Additionally, unique features will be implemented to meet the requirements of the objectives. This includes the automatic analysis of snowline elevation for each aspect, in different requested sub-regions, using masks generated out of the DEM.
Second, Sentinel-2 data offers diverse applications beyond snow classification. For example, forest health monitoring can be conducted to detect bark beetle infestations by analyzing the Normalized Difference Vegetation Index (NDVI), and tracking changes over time. Declines in vegetation health, because of bark beetle activity, can be effectively identified through spectral analysis.
By applying modifications to the classification component, this processing pipeline could also be adapted for alternative applications in various fields, such as forestry.

3.1.- Detection Method used in this Thesis

The snow classification implemented in this algorithm follows a simplified version of the Theia Snow Collection algorithm [2]. The methodology is based on guidance provided in a tutorial by Copernicus Research and User Support (RUS), a service of the European Space Agency (ESA) [14]. This tutorial uses the software SNAP [15] from the European Space Agency to process the satellite images.
The spatial resolution of the generated map is 10×10 meters. However, it is important to note that the Shortwave Infrared (SWIR) band used in the calculation of the Normalized Difference Snow Index (NDSI)[12] originally has a resolution of 20×20 meters. To align it with the higher resolution of the other bands, the SWIR band was resampled using the nearest-neighbor method, effectively dividing each 20-meter pixel into four 10-meter pixels with identical values.
In this thesis the workflow has been implemented in Python scripts to automate the process,

utilizing the Rasterio library for satellite image processing. [1]
In addition, data downloads and statistical analyzes have also been automated.
The chapter 3.3 Workflow and Configuration provides a very detailed explanation of the code workflow.

3.1.1.- Inputs

The user inputs required for the analysis are kept minimal to simplify the workflow. The primary inputs include the coordinates of the area of interest and the observation period's start and end dates, with an option for continuous analysis by setting the end date to the current day.
Additional adjustments can be made for parameters such as classification thresholds, folder locations, log-in credentials for the Copernicus website, and some options related to post processing and statistical analysis. All of these parameters can be set in the configuration file. That file lists all the changeable variables of this program, the file is called `config.yaml` and a more detailed description of this file and the containing variables can be found in Chapter 3.3.1.
Additionally, for generating statistics, a file containing different regions in vector format (.shp) can be incorporated to facilitate the subdivision of data. For example, a single execution of the code could analyze an entire state while still providing detailed regional statistics. This approach aligns with the needs of the avalanche warning service, which operates with smaller microregions within its warning areas. Generating snowline statistics for each of these microregions is essential to enable seamless integration into their existing workflows.
Moreover, the code requires a Digital Elevation Model (DEM) that covers the area of interest. The statistic part of the code is the most accurate if the DEM has the same resolution as the processed image (10x10m). If there are only lower resolutions available, the code will resample the image using the nearest-neighbor method. This method gives the pixel in the new resolution the value of the nearest pixel in the old resolution.
Both the vector file and the DEM must be placed in designated folders during the initial execution of the code. Subsequent runs do not need any additional user input.

3.1.2.- Snowclasification

The classification of snow is done in two steps for better results.
In both steps Snow is classified using the NDSI

$$\mathrm{NDSI} = \frac{\mathrm{Green} - \mathrm{SWIR}}{\mathrm{Green} + \mathrm{SWIR}} \tag{3.1}$$

[1]Rasterio is a Python library designed for reading, writing, and manipulating geospatial raster data, allowing efficient handling of large datasets and integration with geospatial analysis tools.

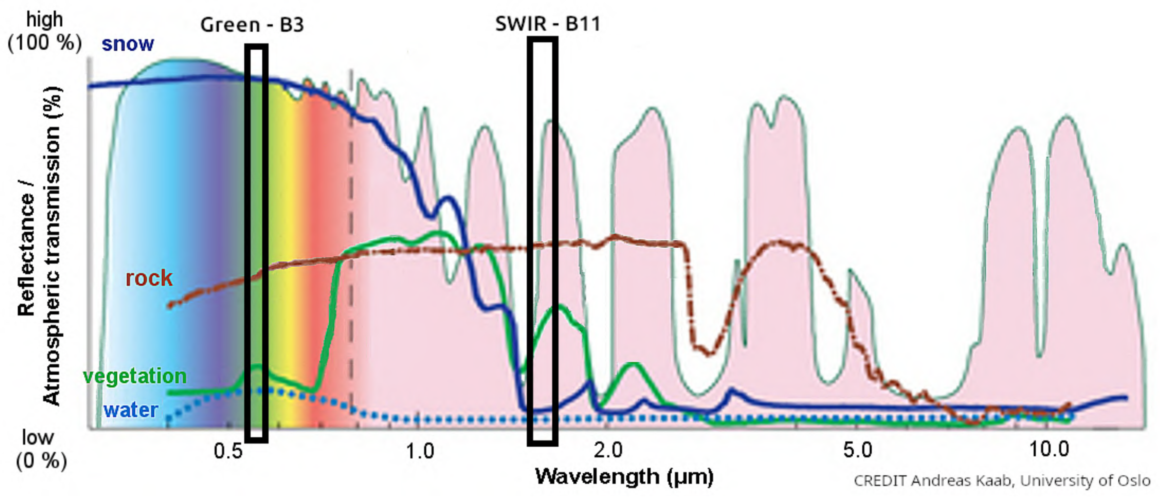


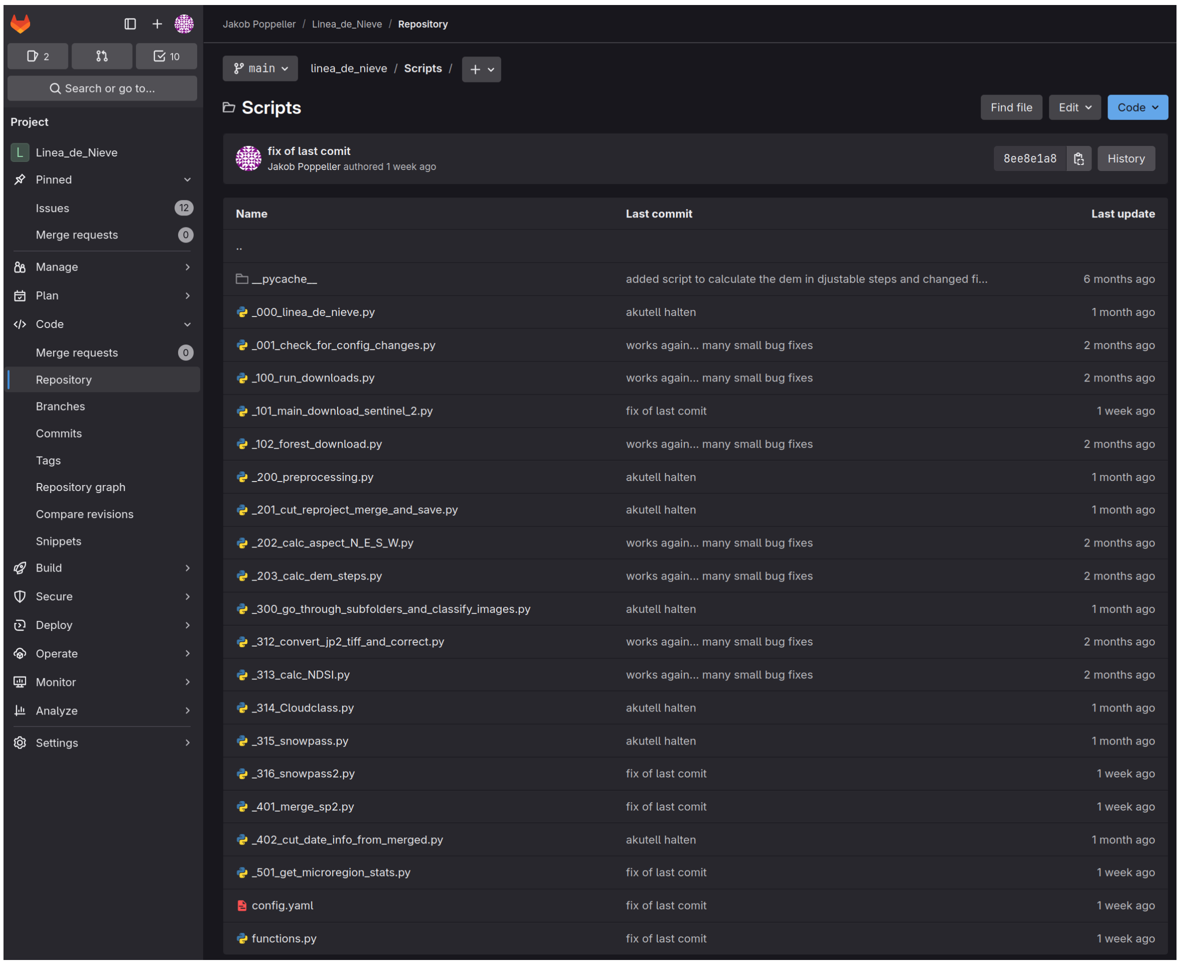
Figure 3: Illustration of the refectance spectrum of snow and the used Bands for the NDSI[5]

and the Red band.
The NDSI indicates a value from -1 to 1 due to the fact that snow surfaces are very bright in the visible spectrum and very dark in the shortwave infrared spectrum. This is visualized in figure 3. Turbid water surfaces like some lakes or rivers may also have a high NDSI value, therefore, an additional condition using the red band is applied to minimize false snow detection in these areas.[2]

In the first step more conservative thresholds for NDSI and red band are applied for the classification. Then a snowline elevation is determined. This is the elevation where a certain percentage of the pixels are classified as snow. All pixels above this snowline are classified again using the same method with less conservative thresholds because it is more likely that a pixel is considered snow in that area. The classification process is shown in the flowchart Figure 6 and described in detail in chapter 3.3.3.

3.2.- Version Control and Documentation

The code developed for this thesis is written in Python, and its progress is documented in GitLab to ensure version control. GitLab is a web-based platform for managing repositories, offering tools for collaboration, issue tracking, and continuous integration in software development projects. A screenshot of this project in Gitlab is shown in Figure 4. Hosting the project on GitLab provides an additional layer of redundancy, safeguarding against data loss caused by local storage failures or accidental user errors.
The project can also be stored locally, with changes on the web interface applied only after



fix of last comit
Jakob Poppeller authored 1 week ago
8ee8e1a8

Name	Last commit	Last update
..		
__pycache__	added script to calculate the dem in djustable steps and changed fi...	6 months ago
_000_linea_de_nieve.py	akutell halten	1 month ago
_001_check_for_config_changes.py	works again... many small bug fixes	2 months ago
_100_run_downloads.py	works again... many small bug fixes	2 months ago
_101_main_download_sentinel_2.py	fix of last comit	1 week ago
_102_forest_download.py	works again... many small bug fixes	2 months ago
_200_preprocessing.py	akutell halten	1 month ago
_201_cut_reproject_merge_and_save.py	akutell halten	1 month ago
_202_calc_aspect_N_E_S_W.py	works again... many small bug fixes	2 months ago
_203_calc_dem_steps.py	works again... many small bug fixes	2 months ago
_300_go_through_subfolders_and_classify_images.py	akutell halten	1 month ago
_312_convert_jp2_tiff_and_correct.py	works again... many small bug fixes	2 months ago
_313_calc_NDSI.py	works again... many small bug fixes	2 months ago
_314_Cloudclass.py	akutell halten	1 month ago
_315_snowpass.py	akutell halten	1 month ago
_316_snowpass2.py	fix of last comit	1 week ago
_401_merge_sp2.py	fix of last comit	1 week ago
_402_cut_date_info_from_merged.py	akutell halten	1 month ago
_501_get_microregion_stats.py	fix of last comit	1 week ago
config.yaml	fix of last comit	1 week ago
functions.py	fix of last comit	1 week ago

Figure 4: Overview of the scripts in the Gitlab reopsitory.

modifications to the code are committed and pushed. Keeping the change history transparent and reproducible.
If the code is already running, a development branch can be created to preserve the original scripts. This allows new versions to be tested before being integrated into the production code, ensuring that the functioning code remains unaffected.
It is also very helpful to add issues of the code directly to the GitLab project, because it makes it easier to keep an overview of what has to be done.
The link to the code of this project can be found in Figure 5

3.3.- Workflow and Configuration

The script is structured into three main sections: Download, Processing, and Postprocessing. The workflow is initiated by executing the script `_000_linea_de_nieve.py`, which sequentially runs all other scripts in the correct order. This is illustrated graphically in the flow-chart

Figure 5: https://gitlab.com/Jakob_Poppeller/linea_de_nieve

6.

A detailed explanation of the functionality and operation of these scripts is provided in the following sections after the explanation the configuration file in chapter 3.3.1.

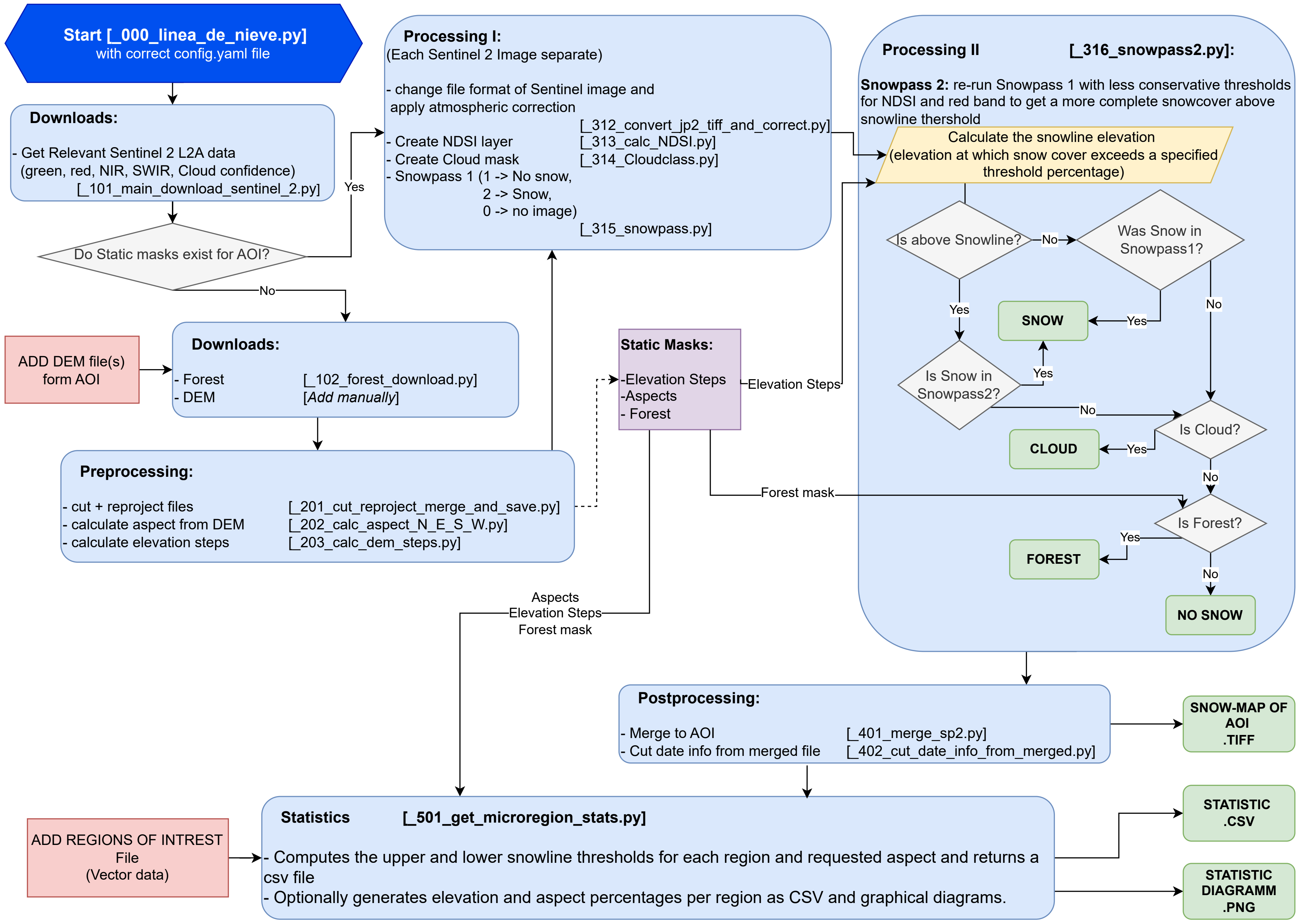

Start [_000_linea_de_nieve.py]
with correct config.yaml file
Downloads:
- Get Relevant Sentinel 2 L2A data
(green, red, NIR, SWIR, Cloud confidence)
[_101_main_download_sentinel_2.py]
Do Static masks exist for AOI?
Yes
No
ADD DEM file(s) form AOI
Downloads:
- Forest [_102_forest_download.py]
- DEM [Add manually]
Preprocessing:
- cut + reproject files [_201_cut_reproject_merge_and_save.py]
- calculate aspect from DEM [_202_calc_aspect_N_E_S_W.py]
- calculate elevation steps [_203_calc_dem_steps.py]
Processing I:
(Each Sentinel 2 Image separate)
- change file format of Sentinel image and apply atmospheric correction [_312_convert_jp2_tiff_and_correct.py]
- Create NDSI layer [_313_calc_NDSI.py]
- Create Cloud mask [_314_Cloudclass.py]
- Snowpass 1 (1 -> No snow, 2 -> Snow, 0 -> no image) [_315_snowpass.py]
Static Masks:
-Elevation Steps
-Aspects
- Forest
Elevation Steps
Forest mask
Aspects
Elevation Steps
Forest mask
Processing II [_316_snowpass2.py]:
Snowpass 2: re-run Snowpass 1 with less conservative thresholds for NDSI and red band to get a more complete snowcover above snowline thershold
Calculate the snowline elevation (elevation at which snow cover exceeds a specified threshold percentage)
Is above Snowline?
No
Was Snow in Snowpass1?
Yes
SNOW
No
Is Snow in Snowpass2?
Yes
No
Is Cloud?
CLOUD
Yes
No
Is Forest?
Yes
FOREST
No
NO SNOW
Postprocessing:
- Merge to AOI [_401_merge_sp2.py]
- Cut date info from merged file [_402_cut_date_info_from_merged.py]
SNOW-MAP OF AOI .TIFF
ADD REGIONS OF INTREST File (Vector data)
Statistics [_501_get_microregion_stats.py]
- Computes the upper and lower snowline thresholds for each region and requested aspect and returns a csv file
- Optionally generates elevation and aspect percentages per region as CSV and graphical diagrams.
STATISTIC .CSV
STATISTIC DIAGRAMM .PNG

Figure 6: Flowchart of the algorithms Workflow.

3.3.1.- Configuration

To configure the code, a `config.yaml` file is provided, where all necessary configurations can be made. These configurations must be set before running the scripts.
The `config.yaml` file includes all adjustable variables used in the code, accompanied by a brief explanation for each variable. This ensures that users can easily modify parameters without directly changing the script.
The configuration file is divided into five sections:

- **1. Downloads:** In this section, it is possible to set a start- and an end-date for the analysis, with the option to set the end-date to 'today'. This feature is particularly useful for the option of continuous analysis, as the code can recognize if the scripts have already been run. If so, it searches for newly available satellite imagery, downloads it, and updates the latest output file with the newly gathered data. Continuous analysis can also be activated in this section.
  Additionally, this section allows the configuration of the bounding box for the analysis. Either preset regions can be selected, or custom coordinates can be defined for the desired region.
  Furthermore, login parameters for `copernicus.eu` can be set in this section to enable access to satellite data. The maximum allowable cloud coverage percentage for downloaded images can also be specified, ensuring that only images with a sufficient amount of usable data are included in the analysis.

- **2. Folders and Files:** This section allows for the configuration of all necessary paths. These include the path to the `download_and_process folder`, where all downloaded and processed files are stored. Additionally, paths to static layer masks and the regions file, which is used to subdivide the area of interest into smaller regions, can also be specified.

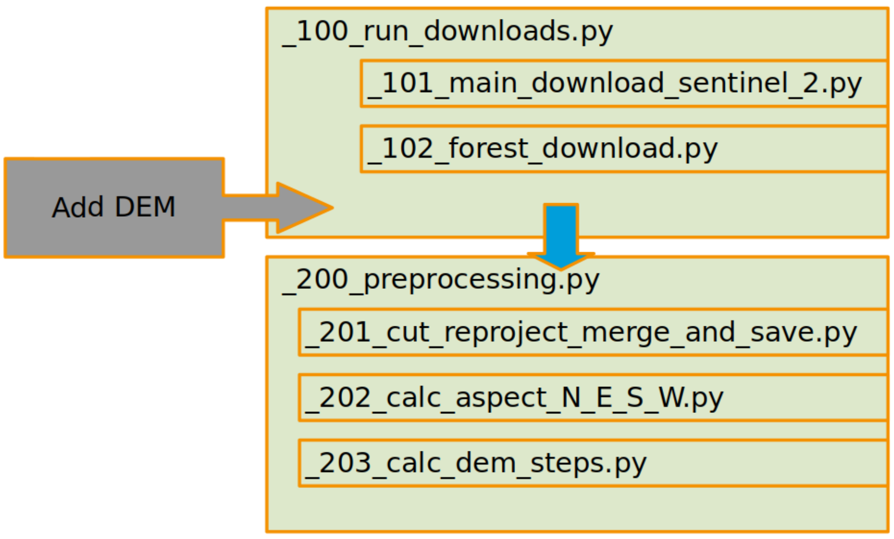
- **3. Processing:** Various aspects of the analysis can be configured in this part. For instance, it is possible to specify whether the folder containing individual satellite images, or parts of it, should be removed after the analysis to conserve disk space.
  Additional settings include the step size of the digital elevation model (DEM) used in the analysis, with a default value of 100 meters, the thresholds for each parameter used in the analysis, and other minor configuration options to adjust the processing workflow.

- **4. Postprocessing and Statistics:** This section contains options for the statistical output of the code.
  For instance, the `number_of_days_to_look_back` parameter allows the specification of how many days before the last downloaded image should be included in the output file and the statistic calculations.
  Different exposition settings can be defined, such as including only north and south exposures, all four exposures, or excluding them entirely from the output statistics in the .CSV file.
  In addition, upper and lower percentage thresholds for snow-covered pixels can be set. The.CSV file will return the lowest elevation with this specified percentage of snow for each region. Furthermore, the `detailed_analysis` option, when activated, generates in addition another .CSV file and a plot of the percentage of snow over the height for each smaller region of the are of interest. This is explained in detail in chapter 3.3.5.

- **5. Debugging:** The final part of the settings includes options that facilitate debugging, such as enabling additional comments to describe the code's actions and infos about the progress of the script while running. There is also an option of saving files from the last run for comparison of diferent settings.

3.3.2.- Downloads and Preprocessing

The first section of the code has the scripts named like `1xx_` or `2xx_` as seen in 7.



_100_run_downloads.py
_101_main_download_sentinel_2.py
_102_forest_download.py
Add DEM
_200_preprocessing.py
_201_cut_reproject_merge_and_save.py
_202_calc_aspect_N_E_S_W.py
_203_calc_dem_steps.py

Figure 7: Overview of the structure from Downloads and Preprocessing

The script `_101_main_downlaod_sentinel_2.py` searches for newly available data from the Copernicus Sentinel 2 program. It constructs the download URL based on the settings and bounds provided in the `config.yaml` file and executes the downloads. The script is designed to download only the required bands for the analysis. It looks also for the latest output product (if `continous_analysis` is set to true in the configuration file) and changes the start date of the files to download accordingly so that only new files that came up since the last run are downloaded.
The following scripts generate static layer masks for the specified area of interest, but only if these files do not already exist from previous runs for the same region. The files are identified based on the coordinates of the bounding box, which are also included in their static filenames.
The script `_102_forest_downlaod.py` downloads the required part of the forest layer from the *EU SCIENCE HUB Global forest cover* if necessary.
The digital elevation model (DEM) must be added manually for any region containing the area of interest[2] prior to the first run of the script. Multiple files can also be placed for use in the construction of the DEM of the area of interest. This is done automatically in the next step.
Once all the files are saved, the script `_201_cut_reproject_merge_and_save.py` reprojects[3] and cuts the downloaded DEM and forest files to the area of interest, which will be saved in a static layer masks folder.
The scripts `_202_calc_aspect_N_E_S_W.py` and `_203_calc_dem_steps.py` use the DEM to calculate two files.
First an aspect file, where each pixel is assigned a value from 1 to 4, representing the main aspects. Inclinations below 15 degrees are filtered out of the analysis to not include any flat areas in this mask.
And the second file, a *stepped elevation* file, that rounds all elevations to a specified step size, as defined in the configuration file (e.g., 100m steps). These files are also saved in the static-layer-masks folder.

3.3.3.- Processing

The order of the scripts in the processing section can be seen in 8. There is also a script called `_300_go_through_subfolders_and_classify_images.py`. This script navigates through all downloaded folders of Sentinel 2 images and runs all the scripts seen in figure 8 in the right order for each of the satellite images. The processing part can also be run alone by just

[2]It can be downloaded for each country from `sonny.4lima.de/`

[3]For the projection, the script calculates the center point of the area of interest and determines the corresponding UTM zone (EPSG:326XX). This approach ensures that the most suitable coordinate reference system is applied consistently.

running this script.

At the beginning of the processing the downloaded image gets converted from .jp2 format

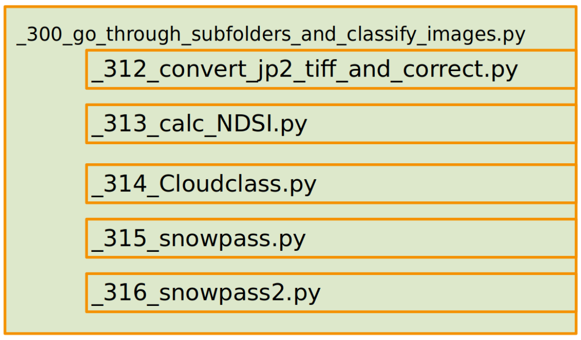

_300_go_through_subfolders_and_classify_images.py
_312_convert_jp2_tiff_and_correct.py
_313_calc_NDSI.py
_314_Cloudclass.py
_315_snowpass.py
_316_snowpass2.py

Figure 8: Overview of the structure from Downloads and Preprocessing

to .tiff format using the script `_312_convert_jp2_tiff_and_correct.py` And in the same step the offset and quantification values are extracted from the metadata and applied to the downloaded images to get the actual physical reflectance values by adding the offset and dividing the result through the quantification value.

With the appropriate reflectance values, the NDSI can be calculated for each image, utilizing the green and shortwave-infrared bands as described in Formula 3.1 in chapter 3.1.2. This is done in the script `_313_calc_NDSI.py`. It calculates the NDSI for the AOI and stores the results in the processing folder, along with the outputs of the following calculations.

The script `_314_Cloudclass.py` generates a cloud mask. The calculations use the quality cloud confidence (QCC) layer[4] along with the bands and the near infrared (NIR). It contains the values 2: Cloud $QCC > 90$, 1: Light Cloud $QCC > 40$ and $NIR > 0.3$ and everything else is 0: No Cloud.

To prevent misclassifications of bright clouds as snow, which was an issue in the development process of this code, a more conservative threshold is applied for now, utilizing only pixels with a cloud mask value of 0 for classification. This ensures that even light clouds are classified as clouds, and snow classification occurs only when the absence of clouds is certain. The application of an already existing cloud detection algorithm for Sentinel 2 imagery would be beneficial. More details on this topic are discussed in chapter 5.3 on future work in this code.

The script `_315_snowpass.py` runs the first step of the snowclassification process by using the NDSI and the red band: The lower thresholds for NDSI is 0.4[5] and for the red band is 0.2[5]. Creating a mask containing three values: 1 - No Snow, 2 - Snow, 0 - No image.

In the second classification step, running `_316_snowpass2.py`, a more detailed classification of the snow takes place. Using the static layer generated earlier, the stepped elevation file

[4]Downloaded automatically together with the different bands from the Copernicus web services.

[5]The values can be adjusted in the configuration file to calibrate the algorithm.

and the just generated snowpass mask. The workflow of this script can also be seen in the flowchart 6 in the 'Processing II' part in the top right corner of the flowchart.
The classified data is combined with the elevation file (100m[5] steps) to determine the lowest elevation threshold where a certain percentage (25 %[5]) of the pixels[6] are classified as snow. This elevation is named the *snowline elevation*.
The second classification step uses less conservative values for the classification: NDSI = 0.15[5] and red band = 0.04[5] and re-runs the `_315_snowpass.py` with these values.
All pixels that have a higher elevation than the *snowline elevation* get their values from the second classification to minimize the gaps in the closed snow surface in areas where it is more likely that there is snow. Pixels below the *snowline elevation* get their values assigned from the first classification step.
All pixels get checked if this pixel is part of the cloudmask. If so it gets classified as Cloud. Then all pixels that are classified as forest in the forest mask and not classified as snow get classified as forest. This ensures that the forest is classified as snow when there is snow visible, but if there is no snow visible it does not get classified as no-snow because of the possibility that there is snow on the ground hidden under the tress from the optical sensor of the Sentinel 2 satellite. Due to this uncertainty, the forest is completely excluded from the statistics, and only open areas will contribute to the analysis.

3.3.4.- Postprocessing

In this section, the script `_401_merge_sp2.py` processes all Snowpass2 files by merging and cropping them to the Area of Interest as defined in the configuration file. The filenames of the resulting files include the date of the most recently classified image as well as the edge coordinates of the boundaries and the used coordinate reference system.
The output is stored in the UINT8 format, allowing each pixel to hold a value between 0 and 255.
For classification purposes, only values between 0 and 4 are utilized: 0 - *No image*, 1 - *No snow*, 2 - *Snow*, 3 - *Forest*, 4 - *Cloud*.
The UINT8 format enables each pixel to contain temporal information as well. The first two digits of each pixel represent the number of days before the merge date that the pixel was classified. If the classification information is older than 25 days, the first two digits are set to 25. The last digit encodes the classification information. An example is shown in Figure 9.
To ensure the maximum of useful data in the merged file the code does not allow to overwrite valid pixels with *cloud* or *nodata* values.
For example, if there is `continous_analysis` activated, in the first run a pixel was classified

[6]excluding the pixels classified as forest

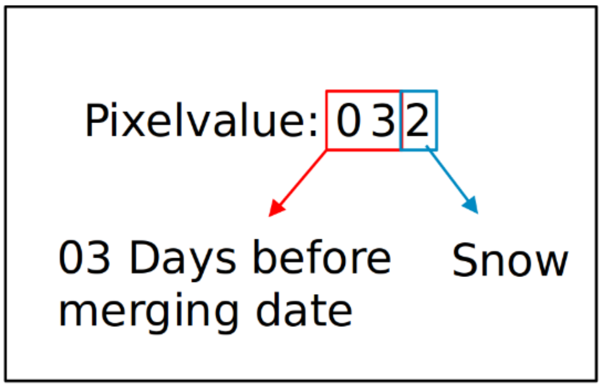


Figure 9: Example of a pixel value in the merged file

as *snow* and merged the value would be `002`. Five days later another image is available and will be downloaded and merged. Here, the same pixel is classified as *cloud*, it will remain snow in the merged data with changed date information: `052`. Indicating that the processing date of this pixel is already five days in the past.
To display the .TIFF file in a GIS program correctly, further modification is required to extract useful data for visualization.
This task is performed by the script `_402_cut_date_info_from_merged.py`. The script utilizes the information provided in the configuration file regarding the number of days to look back at in the analysis. If this information is unavailable or the script is run standalone, user input is requested to specify the desired temporal range for the analysis. Once the script has the information, it deletes all pixels that are older than the defined days to look back and creates a new .TIFF file without the date info printed in the pixel. The filename has the prefix *pixel_date_info_removed_xx_days_before _xxxx-xx-xx*. This file can be opened with an program like GIS program and also be visualized using a fitting layermask.[7]

3.3.5.- Statistics

The final script in the processing pipeline, `_501_get_microregion_stats.py`, is responsible for generating statistics based on the current analysis. Calculations can be performed for smaller subdivisions within the Area of Interest (AOI), referred to as microregions. These microregions must be defined in a vector layer and the path to this file has to be specified in the configuration file.
The code iterates through all microregions. First, it checks if the amount of classified pixels is above a certain threshold to even start the statistical analysis of that region, default is

[7]The layermask used in all the visualisations of the classification in this thesis can be found in the Gitlab repository of the project.

30%[8], to ensure the credibility of the data.
With the help of the static layermasks, the snowline statistics will be calculated.
There are two levels of detail that can be obtained depending on if in the configuration file detailed analysis is set to true or false.
The standard output that always gets created analyzes for each elevation step and for each requested aspect the percentage of snowcovered pixels. Excluding the forest pixels totally from the analysis because the forest floor can be covered in snow even tough the satellite only absorbs the green canopy of the trees.
Then a .CSV file is generated. In that file each line corespondents to one of the microregions. For each region there is a column like North 10% and North 70% and the script writes in each field the first elevation where at least this percentage of snow was found. There are columns for all requested aspects in that format. An example of this file can be seen in chapter 4.2, figure 14
If the detailed analysis is activated, an additional .CSV file for each microregion is generated in a separate statistic folder. This contains more detailed information on the snowline. It shows the percentage of snow in open areas without cloud coverage for each elevation step and aspect of the region.
Additionally this data will be also plotted in a diagram where the x-axis shows the altitude and y-axis displays the percentage of snowcover. For each requested aspect, a graph will be plotted into the diagram to visualize the snowcover in that region. Examples can be seen in chapter 4.2.

3.4.- Validation using Webcam Images

To validate the snow detection process, snow classification results were cross-referenced with manual observations derived from webcam imagery. Six different webcams[9] were selected for this purpose, five in Tirol, Austria, and one in León, Spain. The locations of the webcams in Tirol can be seen in figure 10, the Webcam in León is situated in the skiing resort San Isidro at the Cebolledo station.

Validation samples were collected during the winter season of 2023–2024 [10].
Using the Copernicus web browser, all dates with cloud cover below 40% were identified and listed. Subsequently, five dates were randomly selected from this list.[11]
For each validation date, 5 to 10 control points were identified based on the availability

[8]The threshold can be adjusted in the configuration file
[9]Webcam data for validation were obtained from `www.foto-webcam.eu` and `www.infonieve.es`.
[10]The Winter Season was defined between 01.11.2023 and 30.04.2024
[11]An online random number generator was used: `www.calculator.net/random-number-generator`

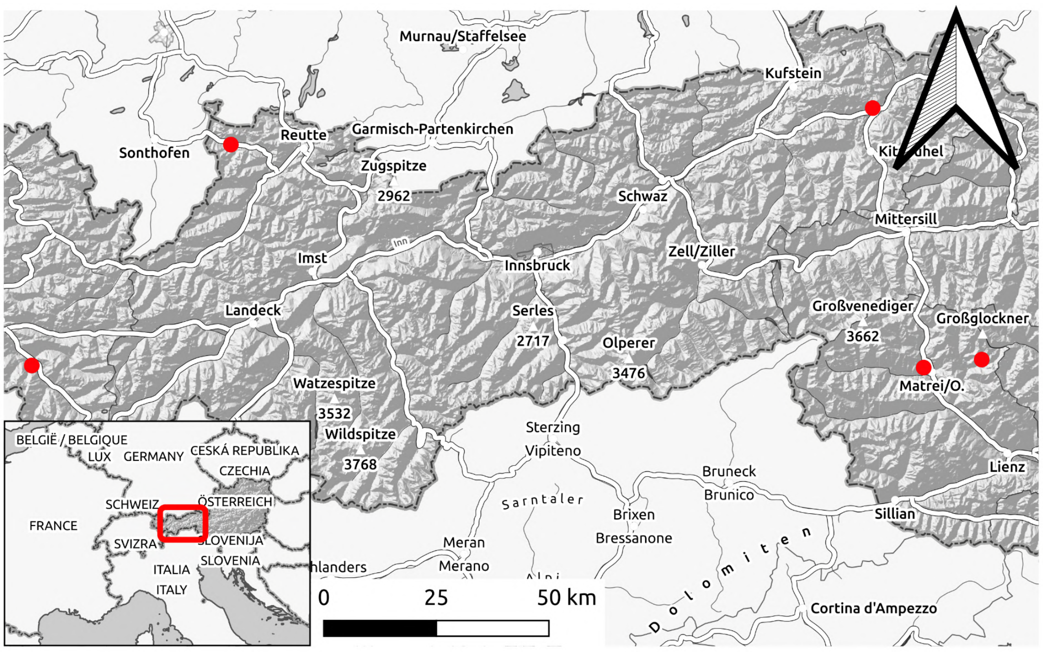


Figure 10: Overview or the Webcam positions in Austria

of matching points in each webcam image and the satellite underlay of the analysis. Two examples can be found in Figure 11 and Figure 12. These points were marked on a mask layer using GIMP[16] and subsequently represented as a point layer in QGIS[17], resulting in a total of 38 unique control points across all regions. Each individual point was classified as either ’Snow’ or ’Not Snow’. In some cases, for example when the control point was on the edge between a Snow and no snow or when the webcam image showed only chunks of snow on the control point, there is also the option for Fractional Snowcover. The results of this analysis can be found in chapter 4.1

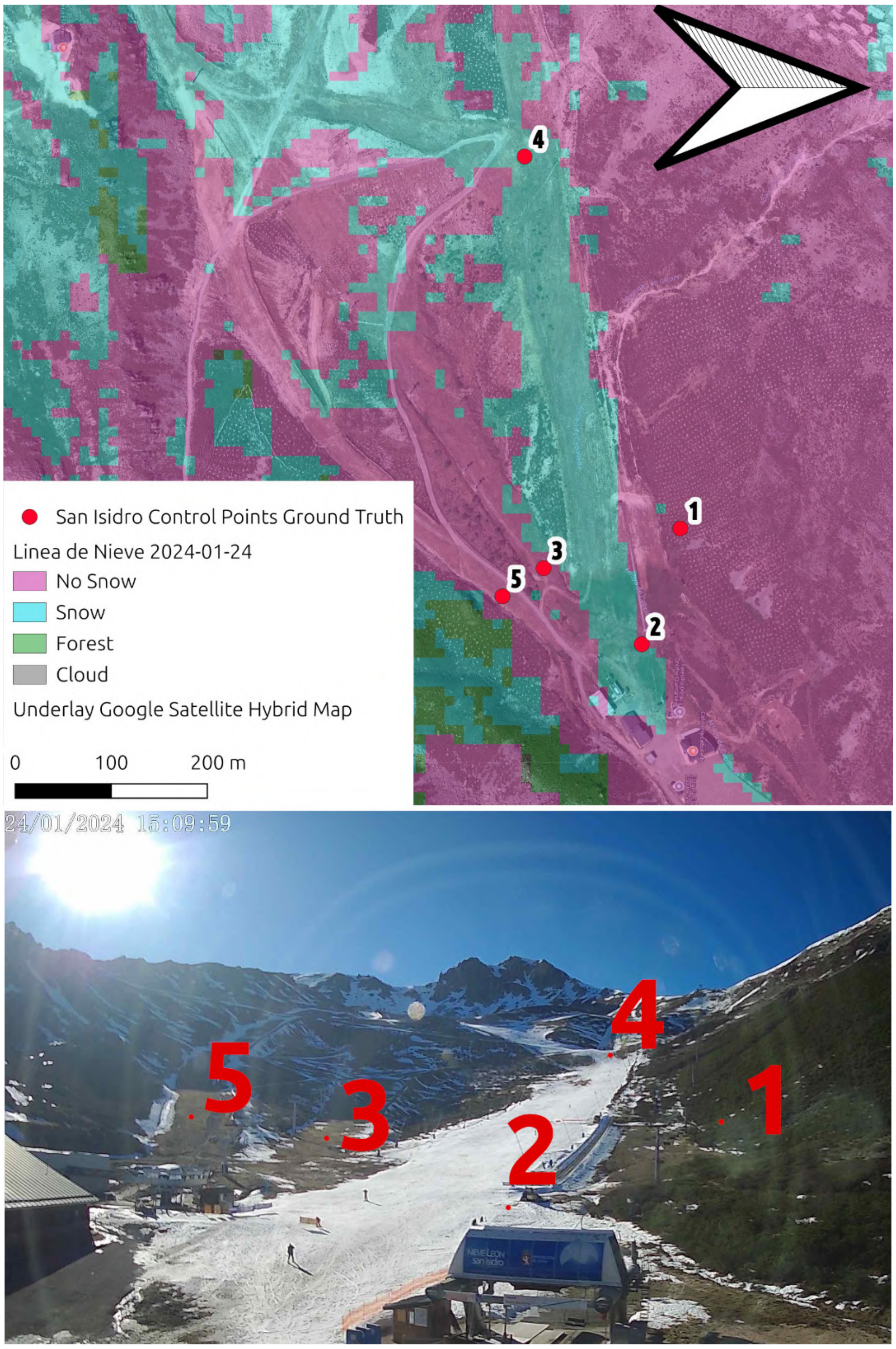

4
1
3
5
2
● San Isidro Control Points Ground Truth
Linea de Nieve 2024-01-24
No Snow
Snow
Forest
Cloud
Underlay Google Satellite Hybrid Map
0 100 200 m
24/01/2024 15:09:59
5
3
2
4
1

Figure 11: Example of the Controlpoints and Groundtruth Analysis San Isidro - León

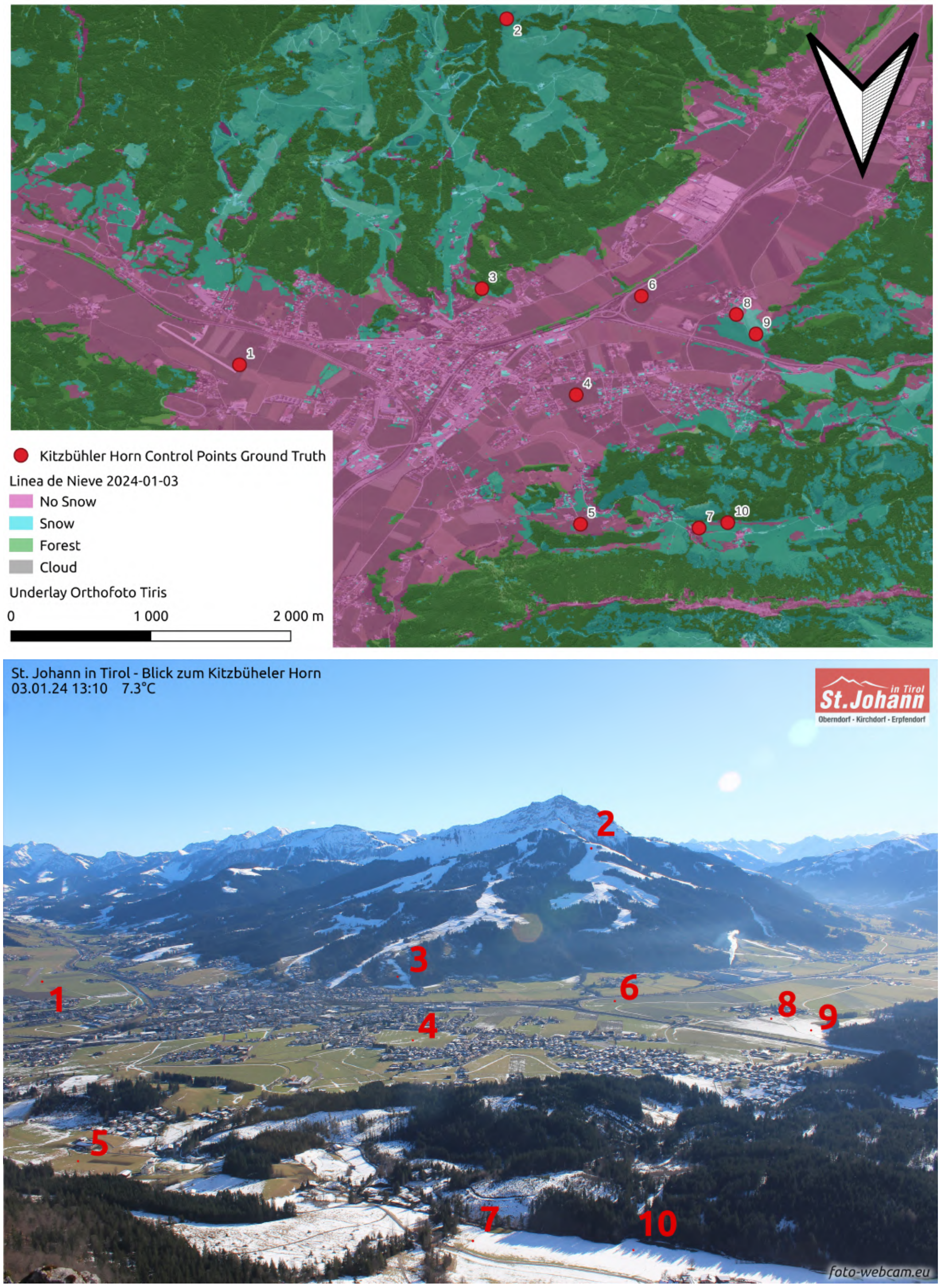

2
3
6
8
9
1
4
5
7
10
● Kitzbühler Horn Control Points Ground Truth
Linea de Nieve 2024-01-03
🟪 No Snow
🟦 Snow
🟩 Forest
⬜ Cloud
Underlay Orthofoto Tiris
0
1 000
2 000 m
St. Johann in Tirol - Blick zum Kitzbüheler Horn
03.01.24 13:10 7.3°C
St. Johann in Tirol
Oberndorf - Kirchdorf - Erpfendorf
2
3
1
6
8
9
4
5
7
10
foto-webcam.eu

Figure 12: Example of the Controlpoints and Groundtruth Analysis Kitzbühler Horn - Tirol

4. Results and Discussion

4.1.- Results of the Groundtruth analysis

For each of the control areas covered by a webcam, five diferent dates of the season 2023-2024 where analyzed for each position, resulting in a total sample size of *N=190* validation points. The thresholds for the analysis are set as mentioned in chapter 3.1.2. Due to the subjective nature of interpreting fractional snow cover, it has been excluded from the accuracy assessment. Consequently, only the colored fields in Table 4.1 were considered. Green fields indicate correct classifications, while red fields represent incorrect classifications.

	Analysed Sentinel 2 Data					
	N=190	Snow	Not Snow	Fractional Snowcover	Cloud	No Data
Webcam	Snow	63	3	0	1	0
	Not Snow	0	93	0	5	0
	Fractional Snowcover	2	6	1	3	0
	Cloud	3	2	0	2	0
	No Data	0	5	1	0	0

Table 4.1: Results of ground truth analysis.

Among the 190 samples analyzed, 165 were definitively classified as either snow or no snow. Of these, 98.1% were correctly classified, demonstrating a high degree of accuracy, while 1.9% were misclassified. 4.7% of the control points were situated on the boundary between snow and no snow. These cases were not included in the validation, as it is acceptable for these images to be classified as either snow or no snow due to the natural uncertainty in these boundary areas. Furthermore, 8.4% of the points were obscured by clouds, either in the webcam imagery or during the analysis process, limiting the ability to make a definitive classification. Another 3.2% of the data were missing due to the absence of webcam imagery for a specific day in one of the regions.
The results obtained from this analysis are consistent with the validation data, which further supports the reliability and robustness of the approach used.
The misclassified pixels were investigated further, revealing that the errors were caused by the presence of thick cloud shadows. While the reflectance beneath thin cloud shadows remains relatively high, allowing the NDSI to be applied effectively, this is not the case under thick cloud shadows, where snow becomes difficult to identify.[18] This issue is examined in more detail in Chapter 4.3, with potential solutions proposed in Chapter 5.3 Future Work.

4.2.- Discussion of the Outputdata

An analysis of the day 17.01.2025 of the region of Tirol, has been done, to discuss the different output data. The microregion of AT-07-04-01_Westliches Karwendel was selected for a closer analysis of the results that can be seen marked in yellow in the overview map Figure 13. Figure 14 shows the Output file from the statistic analysis. The entry of the study area was also highlighted in yellow after the analysis for a better overview.

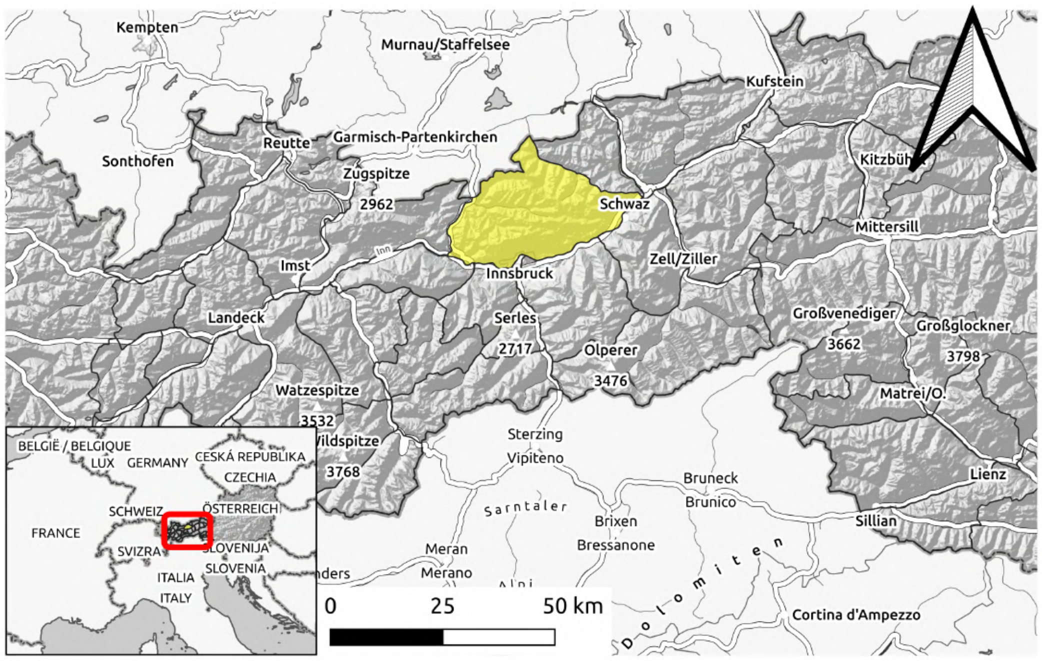
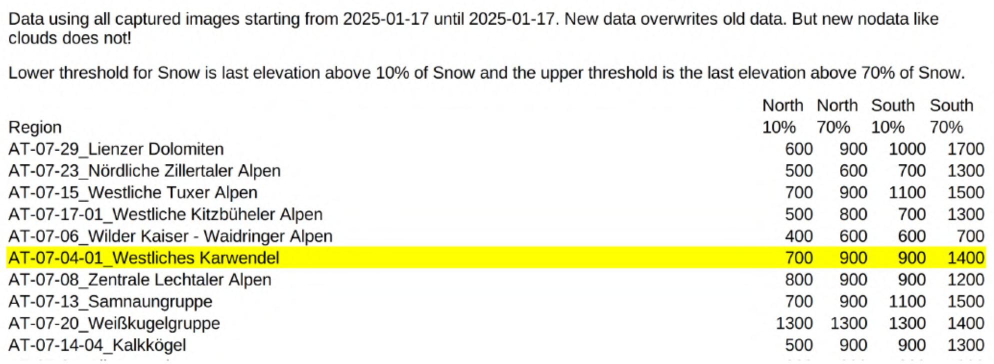


Figure 13: Overview of the analyzed area



Data using all captured images starting from 2025-01-17 until 2025-01-17. New data overwrites old data. But new nodata like clouds does not!

Lower threshold for Snow is last elevation above 10% of Snow and the upper threshold is the last elevation above 70% of Snow.

	North	North	South	South
Region	10%	70%	10%	70%
AT-07-29_Lienzer Dolomiten	600	900	1000	1700
AT-07-23_Nördliche Zillertaler Alpen	500	600	700	1300
AT-07-15_Westliche Tuxer Alpen	700	900	1100	1500
AT-07-17-01_Westliche Kitzbüheler Alpen	500	800	700	1300
AT-07-06_Wilder Kaiser - Waidringer Alpen	400	600	600	700
AT-07-04-01_Westliches Karwendel	700	900	900	1400
AT-07-08_Zentrale Lechtaler Alpen	800	900	900	1200
AT-07-13_Samnaungruppe	700	900	1100	1500
AT-07-20_Weißkugelgruppe	1300	1300	1300	1400
AT-07-14-04_Kalkkögel	500	900	900	1300

Figure 14: Example of the Results Statistics File with one example region highlighted

Within the selected area lies a prominent south-facing ridge, the Nordkette above Innsbruck, which serves as an exemplary case to demonstrate the 10% and 70% snow cover thresholds for the south-facing aspects. The aspect layer from the analysis was used to highlight the south facing areas on the map to get a good overview of the region, as depicted in

Figure 15 the first map. The second map illustrates the .tiff file of the output, styled with a custom layer design. The corresponding layer style, along with the code used for this analysis, is accessible in the GitLab project.[1]
Both maps highlight the elevations corresponding to the snow cover thresholds, with 10% marked at 900 meters in blue and 70% at 1400 meters in red. Additionally, the 700-meter line is displayed in black to represent the lower threshold for snow levels in north-facing areas. while coincidentally the 70% value also aligns with the blue 900-meter line. Remembering that forest is excluded from the statistical analysis, both the 10% and the 70% threshold appears qualitatively accurate.

Figure 16 presents the graphical output of the *detailed_analysis*, where the steepness of the curve reflects the sharpness of the snowline. A vertical ascent like the blue curve of the north sector indicates a sudden transition from *no-snow* to *snow* within a single elevation step. In contrast, the graph for the south-facing areas shows a gradual rise, transitioning from *no snow* to predominantly snow-covered conditions.
A potential explanation for the difference in the steepness of the north- and south-facing curves in this case lies in the varying exposure to sunlight. On north-facing slopes, the absence of direct solar radiation results in slower snowmelt, resulting in an still intact snow-cover reaching into lower elevations. Also, rainfall below a certain elevation accelerates snowmelt, leading to a more abrupt transition between snow-covered and snow-free areas, thereby creating a sharp boundary in the snowline. On the other hand, the south-facing slopes experience more uniform melting as a result of their increased solar exposure. However, snow accumulation persists in avalanche pathways, resulting in a more fractional snow cover across a broader elevation range. These pathways can be seen in Figure 17, the webcam footage of the same day as the analysis.

[1]The link to the project can be found in chapter 3.2.

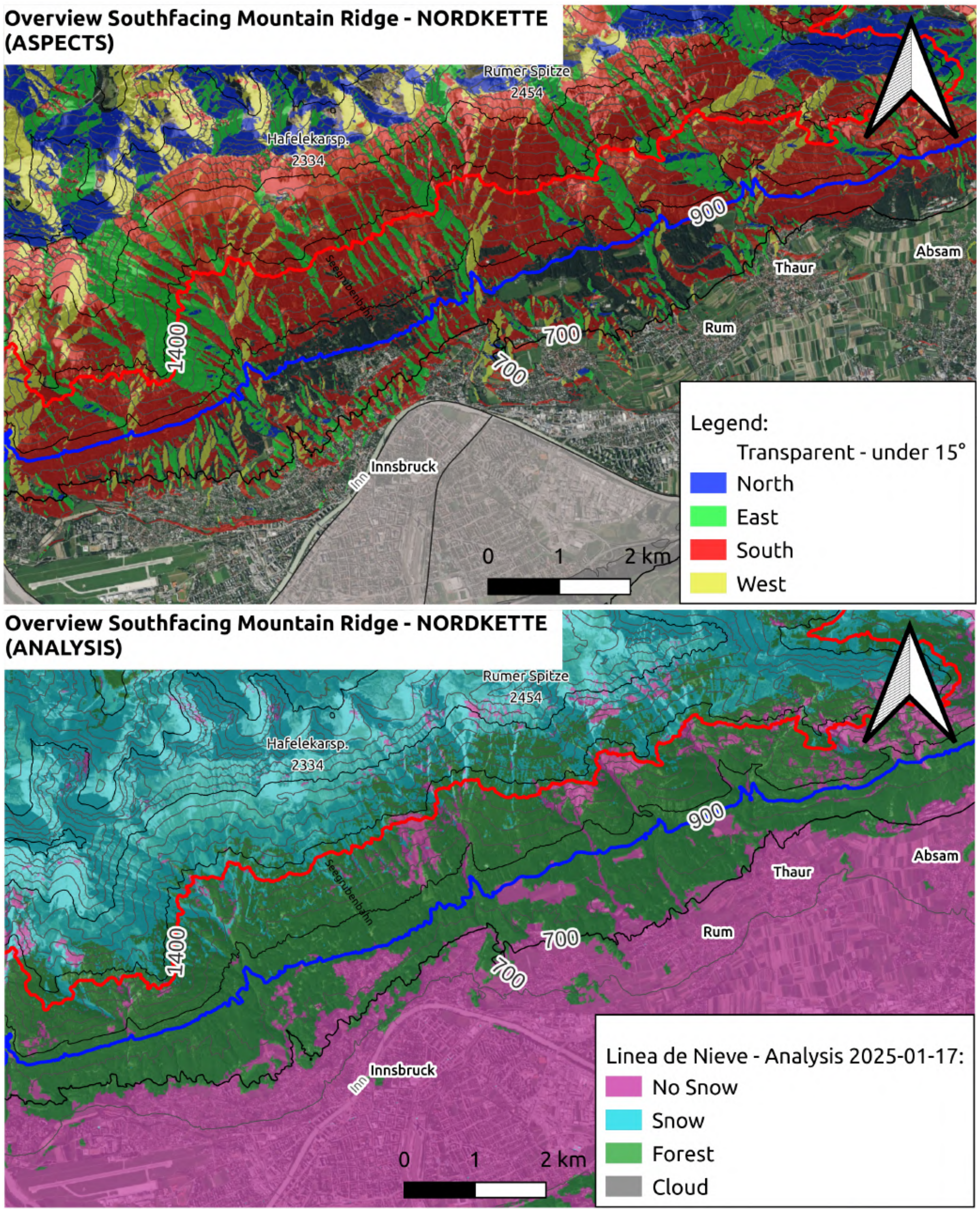


Figure 15: Overview of the aspects and analysis of the Nordkette (Region AT-07-04-01_Westliches Karwendel)

4.3.- Limitations and Challenges

The missclassification of *snow* as *no snow* due to a thick cloud shadow. This is a known issue for the optical analysis of satellite data. During ground truth validation, a textbook example of wrong cloud shadow detection occurred. This is addressed here in more detail here to show the importance for the implementation of a better cloud and cloud-shadow detection algorithm.
The classification of snow with the NDSI (3.1) includes the green band and the short wave infrared band. However, cloud shadows result in reduced reflectance in these bands, leading

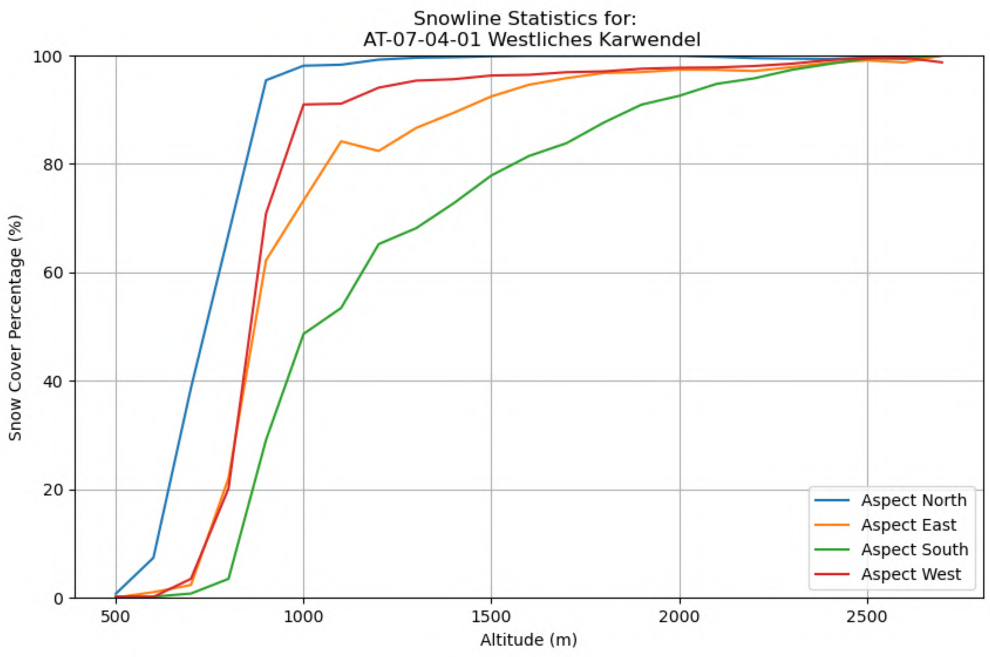


Figure 16: Diagram of the distribution of the Snow in the study area

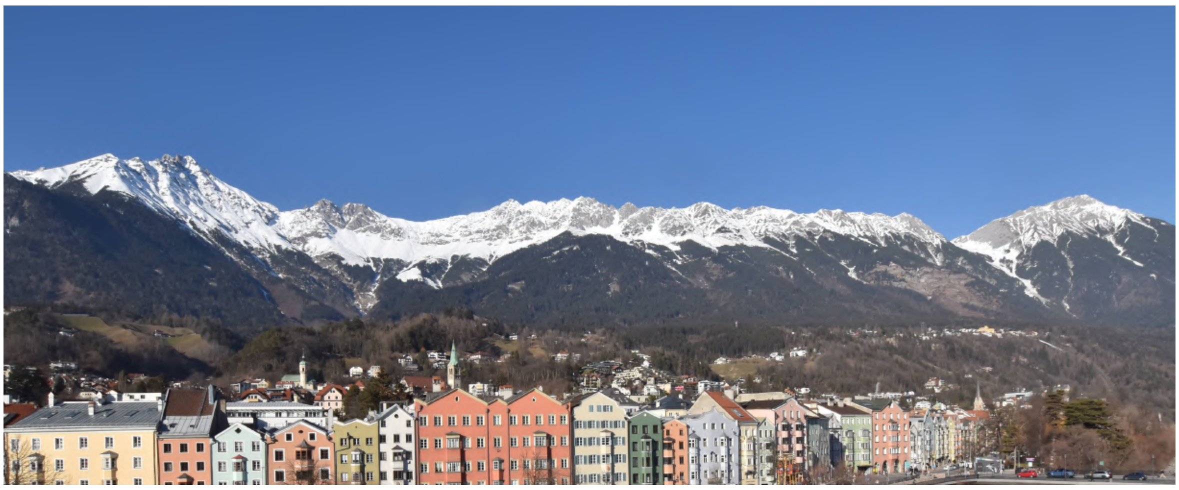


Figure 17: Webcam footage of the study area with avalanche pathways

to darker regions in the imagery.[19]. This is illustrated in 18. In the B3 - green band, the area covered by the cloud shadow is still visible in the bottom left corner of the image. But in B11 - SWIR band this area is not visible. This results in missclasification with the NDSI. That can be seen in the image of the wrong classification. Turquoise represents snow, pink the absence of snow and green shows forest. The webcam footage shows the shadow. The webcam image was taken from the bottom left corner of the images observing the area in the direction of the top right corner. This possible solutions for this issue are mentioned in chapter 5.3 on future work.

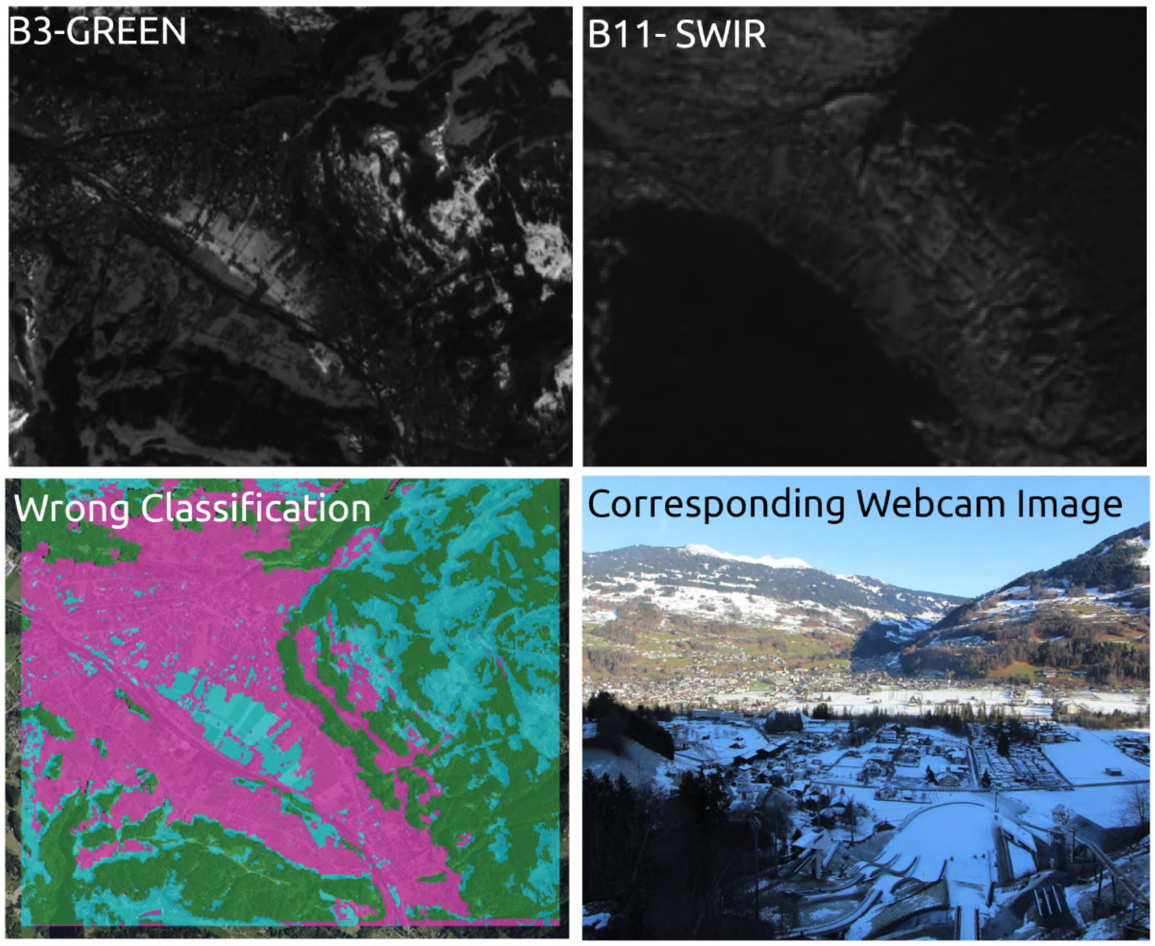


Figure 18: Overview of the wrongly classified points due to Cloud-shadow

Another limitation is that the temporal resolution of 5 days, combined with the issue that the optical analysis can only take place in areas without cloud coverage. It can take several days to get updated information on the snowcover.
This issue is adressed in this algorithm by allowing the definition of a maximum of days to look back for the output images. If this variable is set for example to 14 days, the statistics don´t use older data and also the output map shows nodata in areas that have not been updated for too long. Resulting to be able to guarantee a certain up-to-dateness of the data.

5. Conclusion

5.1.- Summary of the developed Code

This thesis has addressed the challenge of snow classification from Sentinel-2 data. The developed program code provides a simple-to-operate solution for snow classification with high spatial resolution. In addition to the classification of snow cover on a map, it offers more features: For example, the generation of valuable statistical parameters, including the distribution of snow across different aspects and elevations. It is also possible to subdivide the area of the analysis into many smaller regions.
The outputs are provided in universal formats, including the classified maps as .GeoTIFF files, statistical data in .CSV format, and visualizations of snow distribution by elevation and aspect as .PNG files.
The code also includes a configuration file that allows users to adapt settings without modifying the code itself.
It is freely accessible on GitLab, a link to the project can be found in Chapter 3.2.
Designed for integration into existing workflows, the program can be operated on a daily basis to update snow cover maps without reclassifying everything again, which saves time and resources. It can be even implemented in automatic workflows without user interaction.
This tool is particularly useful for the Avalanche Warning Service. For example, the analysis of snow cover can support the identification of a lower threshold elevation for the avalanche warning area in different regions. If the processed results of the code are published, this would also be a very useful tool for mountaineers to assess the current state of the snow.
Despite its advantages, the code has limitations, notably the obstruction of classification by cloud coverage, which can impact accuracy in regions with frequent cloud presence.

5.2.- Summary of the experimental results

The developed code effectively demonstrated its capability to classify snow, no-snow from Sentinel 2 data. The tool provides detailed statistical analysis, and the results met the expectations. For example, that the snowline calculated by the code is in most of the cases clearly higher on south facing slopes than on north facing ones.14
The maps have a high accuracy. A good example for that can be seen in Figure 19 where there is an image of the rests of a skiing slope in spring with chunks of snow of around 15-20 m in diameter correctly classified as snow.

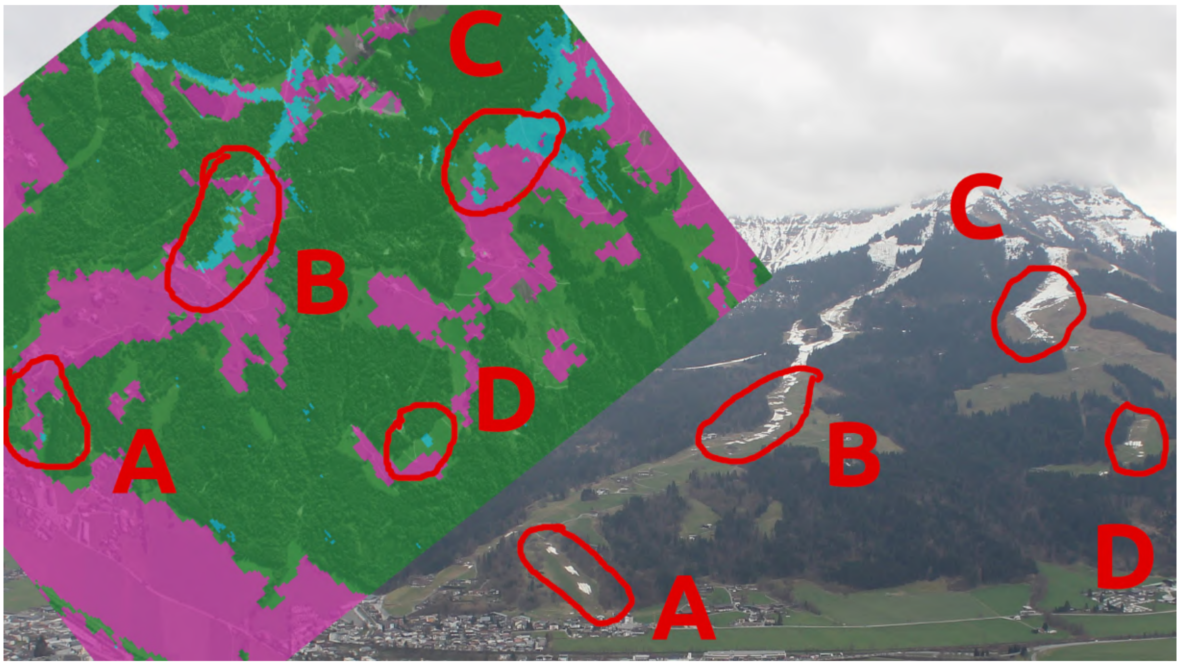


Figure 19: Example for the classification of small areas of snow.
On the left: the analysis (snow - turquoise; no snow - magenta)
On the right: corresponding webcam image.[4]

5.3.- Future Work

Some aspects of the project can be changed to improve its accuracy.
The most urgent improvement should be implementing a more precise cloud and cloud shadow detection algorithm. Several existing solutions could be integrated into the workflow. For example, the CloudS2Mask algorithm employs a deep learning approach to identify clouds and cloud shadows [20]. Another promising option is the F-mask algorithm, which incorporates a global water map to improve the separation of land and water and utilizes a digital elevation model (DEM) to normalize thermal and cirrus bands during the classification process [21]. Enhancing the accuracy of cloud and cloud shadow detection would allow for the application of less conservative thresholds in the cloud mask, thereby reducing cloud coverage in the classified outputs.
Furthermore, the integration of data from additional satellites, for example the MODIS-Satellite[22], into the classification process could improve the temporal resolution. By utilizing different image sources, the frequency of updates could be increased, resulting in higher update frequency of the snow cover maps. However, this approach introduces new challenges, such as the different spatial resolutions of the satellites. Clear rules would need to be established to determine which data should be prioritized in cases of overlap or conflict.
The validation process should be improved to archive more precise results. For example the data could be validated on a spatial area using already existing snow maps and calculating

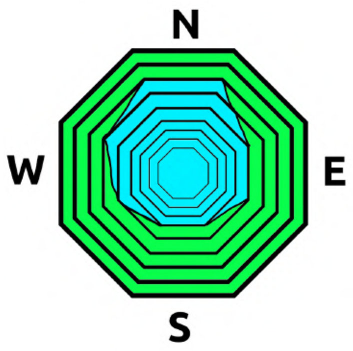


Figure 20: Sketch of a graphical representation of the snowline

the area of overlap and the areas where the analyses differ. With this approach there are not only point samples taken into account and it can be validated on a much larger scale.
Additionally, the statistical analysis process offers significant potential for further development. Depending on the needs of the users, specific improvements can be made. For example, a pictogram of all aspects with the snowline plotted for each region see the sketch Figure 20.

6. Conclusiónes (Síntesis en Español)

El código del programa desarrollado en éste trabajo fin de máster proporciona una solución para crear mapas clasificando la nieve en alta resolución espacial. Además también se generan parámetros estadísticos incluyendo la distribución de la nieve a través de diferentes orientaciónes y para todos los niveles sobre el mar del área del análisis. También es posible subdividir el área del análisis en regiones más pequeñas para obtener una clasificación más precisa.
Los resultados se proporcionan en formatos universales, incluyendo los mapas clasificados como archivos .GeoTIFF, los datos estadísticos en formato .CSV y las visualizaciones de la distribución de la nieve en los niveles de altura y aspecto como archivos .PNG.
El código también incluye un archivo de configuración que permite a los usuarios adaptar los ajustes sin modificar el propio código.
El código es de libre acceso en GitLab.
El programa se puede iniciar diariamente para actualizar los mapas de la cobertura de nieve sin tener que reclasificar todo de nuevo. El código se puede implementar incluso en flujos de trabajo automáticos sin interacción del usuario.
También hay limitaciones, en particular la obstrucción de la clasificación por la cobertura nubosa, que puede afectar a la precisión en regiones frecuentemente nubladas.
También debería implementarse un algoritmo de detección de nubes más preciso, como por ejemplo el algoritmo Cloud2Mask que tiene un enfoque de aprendizaje profundo para identificar nubes y sombras de nubes [20].

Bibliography

[1] R. Mair and P. Nairz, *Lawine: Das Praxis-Handbuch. Die entscheidenden Probleme und Gefahrenmuster erkennen*. Innsbruck, Austria: Tyrolia Verlag, 2015.

[2] S. Gascoin, M. Grizonnet, M. Bouchet, G. Salgues, and O. Hagolle, "Theia snow collection: high-resolution operational snow cover maps from sentinel-2 and landsat-8 data," *Earth System Science Data*, vol. 11, no. 2, pp. 493–514, 2019.

[3] avalanche.report, "Snow height map." https://avalanche.report/weather/map/snow-height, n.d. Accessed: 2025-01-20.

[4] Foto-Webcam, "Foto-webcam: Live webcams." Accessed: 2025-02-01.

[5] E. S. A. (ESA), "Reflectance curves of snow, vegetation, water, and rock." Image retrieved from ESA's multimedia gallery, 2011. Accessed: 2025-01-20.

[6] B. und Steigen Redaktion, "Wie entsteht der lawinenlagebericht? teil 1: Die datenquellen," n.d. Accessed: 2025-01-20.

[7] MeteoSchweiz, "Meteoschweiz publiziert automatisch gemessene schneehöhen auf der webseite," 2024. Accessed: 2025-01-20.

[8] L. Report, "Weather measurements for tirol," 2024. Accessed: 2025-02-01.

[9] Z. für Meteorologie und Geodynamik (ZAMG), "Snowgrid - klimatographien," n.d. Accessed: 2025-01-20.

[10] Z. für Meteorologie und Geodynamik (ZAMG), "Analyse und prognose der schneehöhe für 28 millionen datenpunkte." https://www.zamg.ac.at/cms/de/klima/news/analyse-und-prognose-der-schneehoehe-fuer-28-millionen-datenpunkte, n.d. Accessed: 2025-01-20.

[11] Copernicus Land Monitoring Service, "Fractional snow cover (fsc)," n.d. Accessed: 2024-12-30.

[12] J. Dozier, "Spectral signature of alpine snow cover from the landsat thematic mapper," *Remote sensing of environment*, vol. 28, pp. 9–22, 1989.

[13] Copernicus Land Monitoring Service, "High-Resolution Gap-Filled Fractional Snow Cover (HR FSC)," 2024. Accessed: 2024-12-30.

[14] C. Research and U. S. (RUS), "Copernicus rus training materials cryo03 – snow cover mapping with sentinel-2." `https://eo4society.esa.int/resources/copernicus-rus-training-materials/`, 2022. Accessed: [June 2024].

[15] E. S. A. (ESA), "Snap: Esa sentinel application platform," 2024. Software.

[16] The GIMP Team, "Gimp: Gnu image manipulation program," 2024.

[17] QGIS Development Team, "Qgis: A free and open source geographic information system," 2024.

[18] Y. Zhang, C. Ye, R. Yang, and K. Li, "Reconstructing snow cover under clouds and cloud shadows by combining sentinel-2 and landsat 8 images in a mountainous region," *Remote Sensing*, vol. 16, no. 1, p. 188, 2024.

[19] T. Wang, J. Shi, H. Letu, Y. Ma, X. Li, and Y. Zheng, "Detection and removal of clouds and associated shadows in satellite imagery based on simulated radiance fields," *Journal of Geophysical Research: Atmospheres*, vol. 124, no. 13, pp. 7207–7225, 2019.

[20] N. Wright, J. M. Duncan, J. N. Callow, S. E. Thompson, and R. J. George, "Clouds2mask: A novel deep learning approach for improved cloud and cloud shadow masking in sentinel-2 imagery," *Remote Sensing of Environment*, vol. 306, p. 114122, 2024.

[21] S. Qiu, Z. Zhu, and B. He, "Fmask 4.0: Improved cloud and cloud shadow detection in landsats 4–8 and sentinel-2 imagery," *Remote Sensing of Environment*, vol. 231, p. 111205, 2019.

[22] N. E. S. D. Systems, "Modis (moderate resolution imaging spectroradiometer)," 2025. Accessed: 2025-01-21.

Appendices

6.0.1.- Code sample and explanation of the `_501_get_microregion_stats.py` script

As an example parts of the statistics script has been selected for explaination the whole script can be found here:

In the beginning of the script all necessary Python packages are imported. Including a functions file that is part of the codes and contains some functions that are used in various scripts of this project. Followed by a brief description on how the script works.
Then there are some more functions defined. But to make it easier to follow these functions will be described when called. Therefore the description starts with the main function.

```python
def main(stats_run_as_standalone = False):
    config = functions.load_config()
    root_folder = config['download_and_processfolder']
    file_prefix = config['filename_prefix_for_merged_file_without_pixel_date_info']
    files = functions.list_files_with_prefix(root_folder, file_prefix)
```

First the configuration file gets loaded into the variable config followed by the definition of the path to the files. The function list_files_with_prefix returns a list of all files where the filename starts with a certain prefix out of the requested folder.

```python
    #if activated user interaction is necesary
    if config['ask_which_file_to_analyse'] or stats_run_as_standalone:
        if functions.display_files(files):
            snow_raster_path = functions.choose_file(files)
            print(f"You have selected: {snow_raster_path}")
        else:
            print('\n\nNo merged files found. Script stops. Create a merged File first!')
```

```
389          time.sleep(5)
390          quit()
```

In line 383 the if statement checks if ask_which_file_to_analyse was set to true in the configuration. If so the code proceeds with a user interaction, where it lists all available files, defined in line 374. To choose which file should be analyzed. If there are no files the script stops.
The second option for user interaction is when the script was run directly and not in the process pipeline. Then the variable stats_run_as_standalone is set to true. This is because the `if __name__ == "__main__":` runs the main with that variable set to true. Only if the script is run on its own this variable is true. Otherwise it is defined as false (line 370)

```
391  #else latest date with will be choosen
392  else:
393      latest_date= None
394      for file in files:
395          amount_of_days_looked_back = file.split('/')[-1].split('_'
)[-13] #extracts how many days looked back info
396          if amount_of_days_looked_back != config['
number_of_days_to_look_back']:
397              continue
398          else:
399              file_date=file.split('/')[-1].split('_')[-10] #
extreacts date
400              if latest_date == None or latest_date < file_date:
401                  latest_date=file_date
402                  snow_raster_path = file
403
404      if latest_date == None:
405          print('\n\nNo valid merged file found. Script stops.\nYou
can set ask_which_file_to_analyse to yes and choose the file on
your own if there is one.')
406          time.sleep(5)
407          quit()
408      else:
409          print('\n\n')
410          print('Following file is being used for analysis of the
snowline:', snow_raster_path)
411          print('\n\n')
```

The else part in line 392, when no user interaction is requested and the script is not run as standalone, extracts from the filenames the processing dates and uses the latest file for the analysis. If no file was found, the code stops and returns an error message. Otherwise an

info which file is beeing processed is printed.
The next part of the code constructs the paths of the static masks and loads the microregions file into the variable microregions. And the id and name column is defined. [1]

```
    #detailed analysis creates an extra folder and for each region a file with stats and a diagramm
    detailed_file_path=None #stays None if there is no detailed analysis
    if config['detailed_analysis']:
        detailed_analysis_path= f'{snow_raster_path.split(".")[0]}_stats'
        if not os.path.exists(detailed_analysis_path):
            os.makedirs(detailed_analysis_path)

    initialize_file_overview(microregions['area_label'], snow_raster_path)
```

Detailed file path gets first set to None so that the variable still exists if there is no detailed analysis requested. This part creates a folder for the detailed analysis if requested with the name of the analyzed file`_stats`. Then the initialize_file_overview function[2] is called. This function creates the .csv file and writes the header with information about the data. It creates also all columns that are possible. Always one column for the lower threshold and one for the upper threshold percentage. And this for each aspect and tow columns for all aspects. After entering the data the empty columns will be deleted again.
Then the microrgions file gets tested and reprojected if necessary to the UTM CRS of the project. This is defined by the function `get_ideal_utm_zone_epsg_for_bounds()` from the function script determines the center of the AOI and calculates the ideal CRS for this point.

```
    for index, region in microregions.iterrows():
        if config['debugging_comments']:
            print('Now processing:',region)
        geom = [region['geometry']]
        #check if geom is in bounds
        geom_in_bounds=functions.is_geometry_in_bounds(geom)
        if not geom_in_bounds:
            print('Regin ', region['area_label'],'is not in Bounds! Continuing with next region!\n')

            if not regions_out_of_bounds:
                regions_out_of_bounds = []

            regions_out_of_bounds.append(region['area_label'])
```

[1]This part is not shown but can be seen in the original file.
[2]This function can be found in line 105 of the script

```
504             continue
505         if geom_in_bounds:
506 
507             with rasterio.open(snow_raster_path) as src_snow:
508                 crs=src_snow.crs
509                 snow_class_region, out_transform = mask(src_snow, geom
    , crop=True)
510                 meta_cropped_snow = src_snow.meta.copy()
511                 meta_cropped_snow.update({
512                     "driver": "GTiff",
513                     "height": snow_class_region.shape[1],
514                     "width": snow_class_region.shape[2],
515                     "transform": out_transform
516                 })
517                 pixel_width, pixel_height = src_snow.res
```

The script starts iterating through all microregions with the for loop. Each of the following steps is executed for each microregion.
In line 492 there is the `config['debugging_comments']` that can be activated to get additional comments and data while the code is running to make it easier to find errors.
In line 497 there is a check if the microregion is within the AOI. If it is out of bounds the region is added to a list and the code proceeds with the next microregion. This list will be printed into the .csv file at the end of the analysis.
If the geometry is in the boundaries the code opens the file of the snowclassification and crops it to the shape of the microregion. This is also done with the aspect elevation and file in the same way.

```
559             if len({snow_class_region.shape[1:], elev_region.shape
    [1:], aspect_region.shape[1:]}) != 1:
560                 height = min(snow_class_region.shape[1], elev_region.
    shape[1], aspect_region.shape[1])
561                 width = min(snow_class_region.shape[2], elev_region.
    shape[2], aspect_region.shape[2])
562                     # Update snow_class_region for the output files
563                 meta_cropped_snow.update({
564                     'crs': crs,
565                     'height': height,
566                     'width': width
567                 })
```

The if len statement checks for the snowclassification and the other masks if they have the same dimensions. If not it is cut to the smallest dimensions of given masks. And the metadata gets updated with the new dimesions. This is done also for the other masks. Now everything is set up for the calculations with the actual data.

```
587            # Set pixels to 1 if they equal snow_val or no_snow_val, otherwise set to 0
588            valid_pixels_mask = np.where((snow_class_region == snow_val) | (snow_class_region == no_snow_val), 1, 0)
589
590            #cropped_snow is only snow or no snow now
591            cropped_snow = np.where(valid_pixels_mask, snow_class_region, nodata_val)
592
593            snow_pixels = np.sum(snow_class_region == snow_val)
594            no_snow_pixels = np.sum(snow_class_region == no_snow_val)
595            cloud_pixels = np.sum(snow_class_region == cloud_val)
596            forest_pixels = np.sum(snow_class_region == forest_val)
597            nodata_pixels = np.sum(snow_class_region == nodata_val)
598            total_pixels = forest_pixels+snow_pixels+no_snow_pixels+cloud_pixels
599            total_pixels_excl_forest = total_pixels-forest_pixels
600
601            # Calculate the percentage of valid pixels
602            # Count the number of valid pixels (snow or no snow)
603            num_valid_pixels = valid_pixels_mask.sum()
604            percentage_valid_pixels_forest = (num_valid_pixels / total_pixels_excl_forest) * 100
```

The valid_pixel_mask sets all classified pixels to 1 and everything else to 0. snow_val and all other values like no snow, cloud, forest are defined at the beginning of the script to allow easy adaption if the values should change in an updated version of the project.
The valid pixels mask gets applied to the classified area to recive a layer with only snow, no snow and nodata values called cropped snow. The Block from line 590 to 596 counts the pixels for each of the categories. To calculate the percentage of classified pixels without the forest in line 604.

```
616            message=f"Percentage of classified pixels in open areas (excluding forests) in {region['area_label']}: {percentage_valid_pixels_forest:.2f}%"
617
618            if percentage_valid_pixels_forest < config['threshold_percentage_of_classified_pixels_to_continue_with_stats']:
```

This information message is later on printed into head of the .csv file of the detailed analysis. If the percentage is below the thershold for classification (line 618) the message gets updated to a warning that there is to little data to continue the statistics for this region. This gets printed into the main .csv file instead of the elevations. And the code proceeds with the next microregion.

```
628             # Ensure that the shapes of the masks match before applying the mask
629             if valid_pixels_mask.shape == elev_region.shape:
630                 # Apply the valid_pixels_mask to the masked_elev
631                 nodata_value_elev=9000
632                 filtered_elevation = np.where(valid_pixels_mask, elev_region, nodata_value_elev)
633 
634             else:
635                 print(f"\n\nShape mismatch: valid_pixels_mask shape is {valid_pixels_mask.shape}, masked_elev shape is {elev_region.shape} ")
636                 quit()
```

If there is enough data classified, the code checks if the shapes of the valid_pixels_mask match with the elevation file. This part is implemented because there where issues regarding matching shapes in the development process. And it is still a good check if everything runs as planned.
Line 631 assigns the nodata value for the elevation file. It is 9000 because it is higher than the highest elevation on earth.

```
638             if config['detailed_analysis']:
639                 unique_altitudes = np.unique(filtered_elevation[filtered_elevation < 8999])
640                 detailed_file_path=os.path.join(detailed_analysis_path,f"{region['area_label']}_stats.csv")
641                 initialize_file_detailed(detailed_file_path,unique_altitudes,region['area_label'],message)
```

This part sets up the .csv file for the detailed analysis. If detailed analysis is requested in the configuration file. unique_altitudes filters all elevation steps for that particular region. Then the filename of the .csv file gets created. Initialize_file_detailed creates the header of the file. Including the printing of the message from line 616 in the head. And the file is saved.

```
643             #diferent options to choose in config No expositions, only north and south or all expositions
644             if config['expositions_to_look_at'] not in [0,1,2]:
645                 print('Check Config file: expositions_to_look_at has wrong value it should be 0,1 or 2 ')
646                 quit()
```

```
647
648         if config['expositions_to_look_at'] == 0:
649             current_exposition = 'All Aspects'
650             Snowline_low, Snowline_high = calc_snowline_stats(snow_class_region, filtered_elevation, current_exposition, detailed_file_path)
651             update_file_overview(snow_raster_path, region['area_label'], current_exposition, Snowline_low, Snowline_high)
652
653         elif config['expositions_to_look_at'] == 1:
654             for aspect_condition, exposition_label in [(1, 'North'), (3, 'South')]:
655                 process_exposition(aspect_condition, exposition_label, snow_class_region, elev_region, nodata_value_elev, aspect_region, detailed_file_path, snow_raster_path, region)
```

In line 644 there is a check if the input in the configuration file was done correctly. Otherwise the code stops with an error message. Then (line 648) is a check if the input was 0 which means to look at all aspects without distinguishing between different ones. With this information the clac_snowline_stats function is called.
If there is more diferent aspects to look at, like in line 652 the process_exposition function is called for each aspect. This function cuts the filtered_elevation mask to the curent aspect, and runs the calc_snowline_stats and update_file_overview as seen in lines 649 and 650 in the loop for each aspect.
The next part shows the function calc_snowline_stats.

```
def calc_snowline_stats(filtered_snow, modified_elevation, current_exposition, detailed_file_path):
    config = functions.load_config()
    threshold_value_low=config['lower_threshold_classification']
    threshold_value_high=config['upper_threshold_classification']

    #nodata vlaue for elevation files (higher than the highest point possible)
    nodata_value_elev = 9000
```

First the function is called with following variables:
filtered_snow is the classification file cut to the current microregion. It is called snow_class_region in the main function.
modified_elevation is the elevation mask but only with values on pixels that have been classified as snow or no snow and that are in the right aspect.
current_exposition contains the current aspect as a string.

detailed_file_path contains the path to the .csv from the detailed analysis. If requested, otherwise it is set to None.
In the following lines the threshold values for the lower and upper boundary are defined, from the configuration file and the nodata value for the elevation is defined again.

```
37     #Creates elveation map for snow or no snow pixels
38     filtered_elevation_all = np.where(np.logical_or(filtered_snow ==
    1, filtered_snow == 2), modified_elevation, nodata_value_elev)
39     #Creates elevation map for only snow pixels
40     filtered_elevation_snow = np.where(filtered_snow == 2,
    modified_elevation, nodata_value_elev)
41
42     #writes all unique values down
43     # but exludes everything above 8999 (9000 is used as elevation
    nodata value)
44     unique_values = np.unique(filtered_elevation_all[
    filtered_elevation_all < 8999])
```

filtered_elevation_all creates an elevation mask with only values on pixels that are classified as snow or no snow in the right aspect.
filtered_elevation_snow does the same but only for pixels classified as snow.

```
43     #all unique elevations, excluding everything above 8999 (9000 is
    used as elevation nodata value)
44     unique_values = np.unique(filtered_elevation_all[
    filtered_elevation_all < 8999])
45
46     #for exposition in cropped_aspect:
47     percentages = {}
48
49     for value in unique_values:
50         # Create a mask where inputfile_mask equals the current value
    and input_snowpass1 snow or no snow but not nodata
51         total_pixels_each_elev = np.sum(filtered_elevation_all ==
    value)
52
53     # Count pixels where both count_value and input_snowpass1 == 2 are
     true
54         count_snow = np.sum(filtered_elevation_snow == value)
```

Line 44 creates a list of all unique elevations that contain classified values in that given aspect. The for loop stats iterating through all elevation steps and counts in line 51 all classified pixels in that step and in line 54 it counts all snow pixels in that same step.

```
56         if total_pixels_each_elev > 0:
57             percentage = (count_snow / total_pixels_each_elev) * 100
```

```
58         else:
59             percentage = 0.0  # I case of division by zero
60 
61         percentages[value] = percentage
62     if config['detailed_analysis']:
63         update_file_detailed(detailed_file_path, current_exposition,
   percentages)
```

In this part the percentage of the snowcover is caclulated and if requested printed for that elevation step in the .csv file.
After that the loop ends. So there is now a percentage of snow for each elevation step.

```
65     sorted_percentages = sorted(percentages.items(), reverse=True)
66     last_value_above_threshold_high = None
67     last_value_above_threshold_low = None
68 
69     for value, percentage in sorted_percentages:
70         if percentage > threshold_value_low:
71             last_value_above_threshold_low = value
72         else:
73             break
74 
75     if last_value_above_threshold_low is not None:
76         print(' ')
77         print(f"The last Elevation on the {current_exposition}-Side
   above {threshold_value_low}% Snowcover is: {
   last_value_above_threshold_low}")
78         print(' ')
79     else:
80         print(f"No elevation found above {threshold_value_low}%
   Snowcover")
81         last_value_above_threshold_low=9000
```

In line 65 the list gets sorted so that the highest elevation is analyzed first.
The for loop assigns now in each step the percentage to the last_value_above_threshold_low, until it is below the threshold_value_low then it breaks the loop because a lower threshold was found.
Line 75 writes a message in the commandline with the info of the percentage.
If no threshold was found, that means mainly that there is not enough snow in that region in given aspect. And the last_value_above_threshold_low is set to the nodata value.
The same for loop is applied for last_value_above_threshold_high to find the upper threshold.

```
98    return last_value_above_threshold_low,
   last_value_above_threshold_high
```

Finally the function returns the high and the low threshold. Which get written in the .csv file as allready seen in line 651.

```
662                 if config['detailed_analysis']:
663                     remove_empty_columns_detailed(detailed_file_path)
664                     create_plots_detailed(detailed_file_path)
665
666     remove_empty_columns_overview(snow_raster_path,
    regions_out_of_bounds)
```

At the end of the loop for each microregion the remove_empty_columns_detailed cuts all aspects that did not get any data asigned. And saves a clean .csv file.
After that the diagrams are plotted in line 664. This is the end of the loop that iterated through all microregions. The code proceeds now with the next microregion until it has looped though all regions each one has percentages assigned.
At last also the main .csv file gets all empty columns removed.
At the end the function create_plots_detailed is explained. It uses the python package `matplotlib.pyplot`:

```
320 def create_plots_detailed(detailed_file_path):
321     output_png_path = detailed_file_path.replace(".csv", ".png")
322     region_name = os.path.basename(detailed_file_path).replace(".csv",
        "")
323     region_name = region_name.replace("_stats","").replace("_"," ")
324     with open(detailed_file_path, 'r') as file:
325         lines = file.readlines()
```

First the output path is defined by changing the sufix of the stats file from .csv to .png. Then the regions name is extracted out of the filename.
Line 325 opens the .csv file to extract the data for the diagram.

```
328     header_index = 1
329     data_index = 2
330
331     header = lines[header_index].strip().split(';')
332     data = [line.strip().split(';') for line in lines[data_index:]]
333     #print(data)
334     # Create a DataFrame
335     df = pd.DataFrame(data, columns=header)
336     df = df.apply(
337     lambda col: col.str.replace('%', '').replace('-', 'NaN').astype(
    float) if col.name != "Altitude" else col
338     )
339     df["Altitude"] = df["Altitude"].astype(float)
340
341     # Plot the data
```

```
342     plt.figure(figsize=(10, 6))
343 
344     for column in df.columns[1:]:  # Skip the Altitude column
345         plt.plot(df["Altitude"], df[column], label=column)
```

Then the header is defined and a dataframe is created.
Line 342 defines the size of the diagram. And in the following lines each column, one for each aspect is is plotted.

```
347     # Add labels, title, and legend
348     plt.xlabel("Altitude (m)")
349     plt.ylabel("Snow Cover Percentage (%)")
350     plt.title(f"Snowline Statistics for: \n {region_name}")
351     plt.legend()
352     plt.grid(True)
353     plt.ylim(0, 100)
354 
355     # Save the plot
356     plt.savefig(output_png_path)
357     plt.close()
358     print(f"Plot saved to {output_png_path}")
```

Finally, lables and titel are added and the file is saved.