University of Potsdam

Institute of Environmental Science and Geography

Bachelor Thesis

Examiner: M.Sc. Nina Nesterova

Examiner: Prof. Dr. Guido Grosse

Date of submission: 26.03.2025

Bachelor of Science Geoecology

Topic: Variability of retrogressive thaw slumps across Siberia

By: Julia Heitz
Student ID number: 813339
julia.moritz@uni-potsdam.de
moritzjulia@yahoo.de

Abstract

Permafrost regions are thawing in the face of climate change. Permafrost thaw often entails greenhouse gas emissions and landscape changes within a formerly stable environment. One type of permafrost thaw feature are retrogressive thaw slumps (RTSs). The interest in RTSs is rising, because of their large impact on the surrounding landscape and the climate (Nesterova et al. 2024). To ease the scientific understanding and communication regarding the variability of RTSs, this work analysed different RTS properties. RTS spectral variability was the main focus, with the Pan-Arctic Visualization of Landscape Change (2003-2022) and Validation Dataset 2 as key resources. Five study areas, each 10,000km^2 large, are spread across Siberia. Each of them contains 10 randomly located sub areas of 100km^2. Each sub area contains known RTS outlines. The spectral slope (the average change of spectral index over a specific timeframe) of RTSs was compared between different study areas and, in addition, within the individual study areas. Moreover, single RTSs were classified regarding their terrain position and their morphology. RTSs within the same study areas and the same terrain position or morphology class were also compared regarding their spectral slope. The results show a clear variation regarding the spectral slope of different study areas, without any significant similarity. While, on a smaller spatial scale, different sub areas within the same study area are predominantly heterogeneous, a few significant similarities in spectral slope have been found. And even single RTSs located within the same study area did not have a clear correlation between either terrain position or morphology and their spectral slope similarity. Additionally, hints of a connection between terrain position and RTS morphology have been observed. The majority of RTS within 4 out of 5 areas are located at the terrain position 'lake shore', most of which exhibit a combined morphology including features of both thermocirque and thermoterrace RTS. The key result of this study is that the spectral slope of RTSs is strongly variable, and a RTS similarity classification scheme was developed. This classification scheme has the potential to be modified for the comparison of other RTS properties as well.

Table of Contents

List of Abbreviations

ALD	Active layer detachment slide
ANOVA	Analysis of variance
Approx.	Approximately
Cf.	Confer (compare)
CTL	Cryogenic translational landslide
E.g.	Example given
GCV	Global shoreline vector
I.e.	Id est (that is)
Resp.	Respectively
RTS	Retrogressive thaw slump
TCB	Tasseled cap brightness
TCG	Tasseled cap greenness
TCW	Tasseled cap wetness
TP	Terrain position
Vs.	Versus
WGS	World geodetic system

1 Introduction

The warming of the arctic is evident in hundreds of different changes in the polar landscape. Changes within the polar landscape differ in scale, shape, impact and recognition. This thesis discusses a specific thaw feature that is called *retrogressive thaw slump* (RTS) and only occurs in permafrost regions.

RTSs generally are a type of slope failure (Mackay 1966). RTS formation requires two conditions: the permafrost needs to have a high ground ice content (Nesterova et al. 2024; Mackay 1966) and the ground surface must be at a slope, often hillslope, where gentle slopes of <5° degrees are sufficient (Leibman et al. 2023). An overview of terms used to describe the process of slope failure is described within table 2 of Nesterova et al. 2024.

The first step of RTS formation is called *genesis*. Genesis starts when a trigger event (e.g. wildfires, anthropogenic actions (e.g. mining) or coastal erosion) induces a "Mass-wasting on seasonal ice at the base of the active layer" (Nesterova et al. 2024 p. 4802) (Cryogenic translational landslide (CTL) or Active layer detachment slide (ALD)). If the exposed ground ice is not covered by sediment and stabilized with vegetation regrowth, the ground ice remains exposed to warm air and solar radiation, which induces a "[m]ass-wasting on massive ground ice" (Nesterova et al. 2024, p. 4802) called *ALD* or *cryogenic earthflow*. If this exposed ice is not covered with sediment and stabilized through vegetation regrowth, further melting of massive ground ice (or thawing of ice-rich permafrost) combined with denudational processes result in the formation of concave hollows. The term "massive ground ice" can describe different types of ice: buried glacial ice, thick ice layers, or large syngenetic ice wedges.

These concave hollows manifest in three primary morphologies: *thermocirque*, *thermoterrace* and a *combination of both* (Nesterova et al. 2024). A schematic of RTS formation is presented in Figure 1.

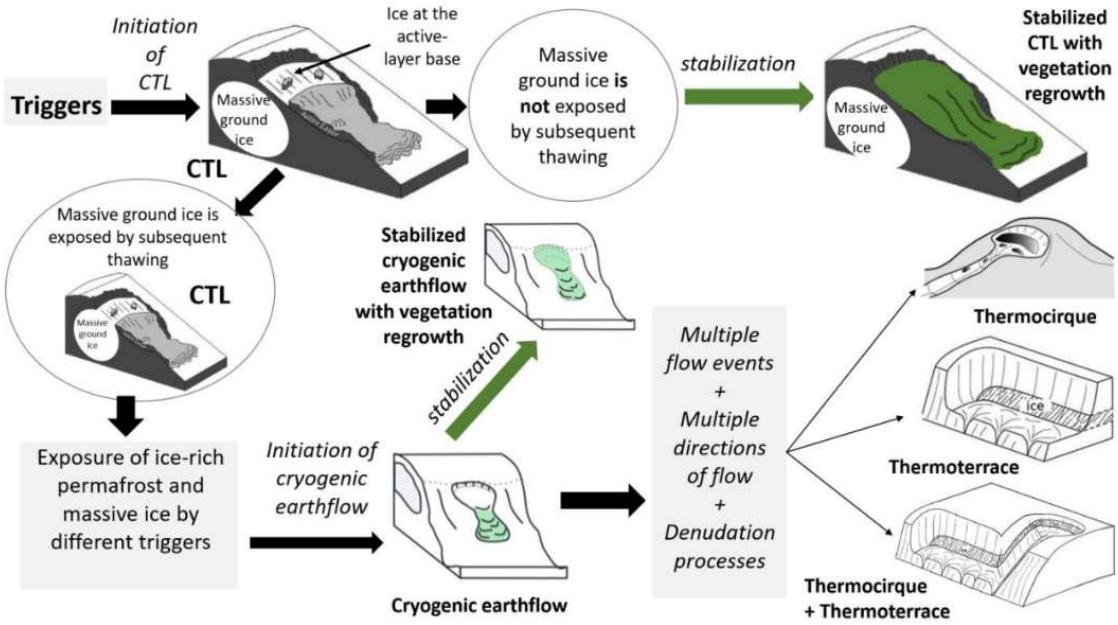


Figure 1: ***Diagram of the theoretical concept of RTS formation and the resulting landforms.*** *Reproduced with permission from Nesterova et al. 2024.*

Thermocirques can be described as amphitheatrical or cirque-like hollows opposed to thermoterraces that have a terrace-like or elongated shape (Nesterova et al. 2024). Thermoterraces often occur at shorelines and result from ice cliff retreat. Thermocirques are typically found at gullies, lake shores and river shores (Nesterova et al. 2024). Both thermocirques and thermoterraces follow the same development scheme, as discussed above. Sometimes the combination of both morphologies can be found.

RTSs have four required morphological features as shown in Figure 2. Several other morphological features only occur in some RTSs depending on their local characteristics. The four essential parts for RTSs are the *headwall* ("A steep retreating wall consisting of ablating ice and frozen sediments at the back of the RTS"), the *slump floor* (the "area of the hollow's bottom", that is low angle to horizontal), the *mudflow* ("The meltwater stream that carries thawed viscous sediment material downslope across and out of the slump floor") and the *edge* ("The boundary line of the headwall or entire landform") (Nesterova et al. 2024 p. 4792).

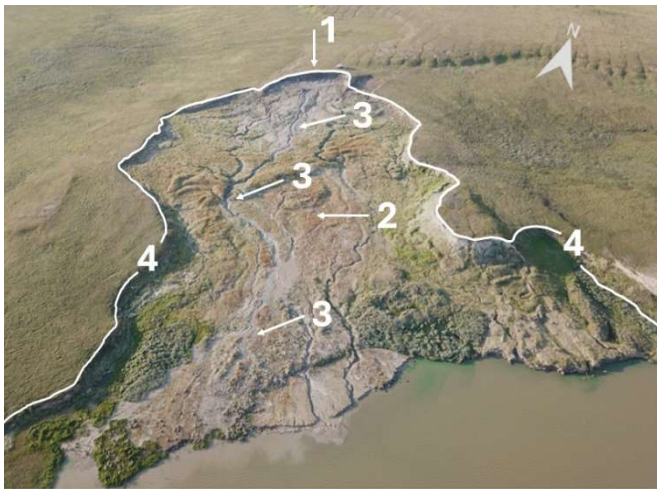


Figure 2: ***Essential Parts of RTSs: (1)*** *Headwall,* ***(2)*** *Slump floor,* ***(3)*** *Mudflow,* ***(4)*** *Edge. Image reproduced with permission from (Nesterova et al. 2024), modified.*

Triggers for RTS formation are saturation of the active layer, fires, bank or coastal erosion, extreme climate events, anthropogenic activities or earthquakes. The two main conditions for the formation of RTSs are (hill-)slope and ground ice content, but many other factors also influence RTS development. To list only a few of them: the aspect and the major relief partly condition the amount of rainfall and absorbed solar radiation and therefore the seasonal thaw depth and water saturation of the active layer, which influences the mass-wasting. The slope angle and the soil cohesion (influenced e.g. by soil micro structure) influence the speed of the mass-movement. The terrain position, defined as direct spatial and physical proximity to a large terrain feature such as lakes or rivers, strongly impacts processes of the RTS development such as abrasion of the lower edge. Other parameters that influence the RTS formation include average ground temperature and type of vegetation. To summarize: the geographical setting strongly impacts RTS formation and development.

RTSs have a high impact on the topography of the landscape, the features can be up to several tens of meters deep and the extent can reach several tens of hectares (Nesterova et al. 2024). The number and the size of these features are rising. Moreover, the disturbance of the vegetation harms the ecosystem and large amounts of sediments, carbon, nutrients and, in some cases, contaminants are mobilized. These changes influence the geochemical fluxes, increase the ratio of released CO_2 and impact the hydrology and water quality (Nesterova et al. 2024). Due to their enormous impacts the research interest in RTSs is rising. One of the challenges is getting an accurate understanding of the highly variable characteristics of RTSs (Nesterova et al. 2024). To

contribute to this important field and to deepen the understanding of RTSs, this work investigates the variability of retrogressive thaw slumps across Siberia.

1.1 Retrogressive thaw slumps in Siberia

Siberia, the largest and most northern part of Russia, surrounds nearly half of the Arctic Ocean. Its vast majority counts as permafrost which is defined as ground that stays at or below 0°C for at least two years in a row (Shur et al. 2011). According to Obu et al. 2018 most of Siberia is underlain by continuous permafrost (90-100 % of the area is permafrost) and according to Jones et al. 2022 a high percentage of the area has a high or medium ground ice content (10% to >20% ice content). The high chance for ground ice and the huge area makes Siberia the perfect place to look into the variability of RTSs, the high ground ice content increases the probability of the appearance of retrogressive thaw slumps and the huge area contains different geographical settings that most probably increase the variability.

This study looks at five different regions (Southern Taymyr, Northern Olenek, Chokurdakh, Iultinsky (Chukotka) and Southern Verkhoyansk Range) that are spread wide across Siberia, visible on the map (Figure 3). Each area is 10,000 km² large and contains 10 randomly located sub-areas that each are 100km² large.

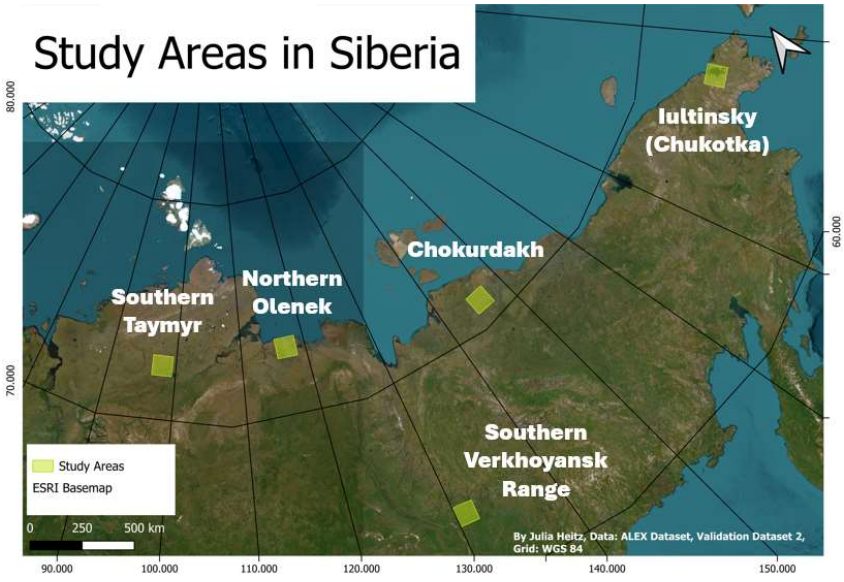


Figure 3: ***Overview of the study areas****. Each yellow square is 10,000 km² large.*

The rising interest in RTSs and the lack of knowledge about their variability in their appearance and characteristics raises questions. What kind of parameters vary between RTSs? Are there patterns? How much do they vary? Are there linear relationships between varying parameters and components of the geographical setting?

Parameters that could be varying and are therefore interesting to look at are e.g. temporal

dynamics (headwall retreat, change in size and numbers, link to air temperature or fires), volume change, spectral variability (spectral slope = average change of spectral index over specific timeframe) and morphometrical parameters (e.g. size, height of headwall, length-width ratio). In the context of this study, the spectral slope appears to be the most promising observable since it combines multiple types of information into three different indices which can clearly identify landscape changes within RTSs. The main hypothesis is supplemented with multiple sub-hypothesis to structure the research.

Main hypothesis	Retrogressive thaw slumps within the study areas show differences in one or more parameters.
Sub-hypothesis 1	The combined spectral slope of the RTSs in one area is never the same as the combined spectral slope of the RTSs in any of the other areas.
Sub-hypothesis 2	The spectral slope of all the RTSs in one selected sub-area is never the same as the spectral slope of all the RTSs in any of the other sub-areas within the same main area.
Sub-hypothesis 3	The spectral slope of single RTSs that are located in a certain terrain position is always the same as the spectral slope of single RTSs that are located at the same terrain position within the same main area.
Sub-hypothesis 4	The spectral slope of single RTSs that have a certain morphology is always the same as the spectral slope of single RTSs that have the same morphology within the same main area.

1.2 Description of the data

Multiple datasets have been combined in this study. While some datasets revealed landscape changes over time, others provided information on features and their location.

1.2.1 Validation Dataset 2

Nina Nesterova (orcid.org/0000-0001-7055-9852) is the contact person for Validation Dataset 2 used in Nitze et al. 2024a. The geographic extent comprises eight different areas of central and eastern Siberia, each site is approx. 10,000km^2 large and contains 10 randomly located 100km^2 squares. Within each square the outlines of all RTSs have been manually mapped, drawn with a 5-meter buffer, by Emma Schütt (overview is shown in Figure 4). Only the areas 1,2,3,4 and 6 contain RTSs and therefore only these areas are included in this study.

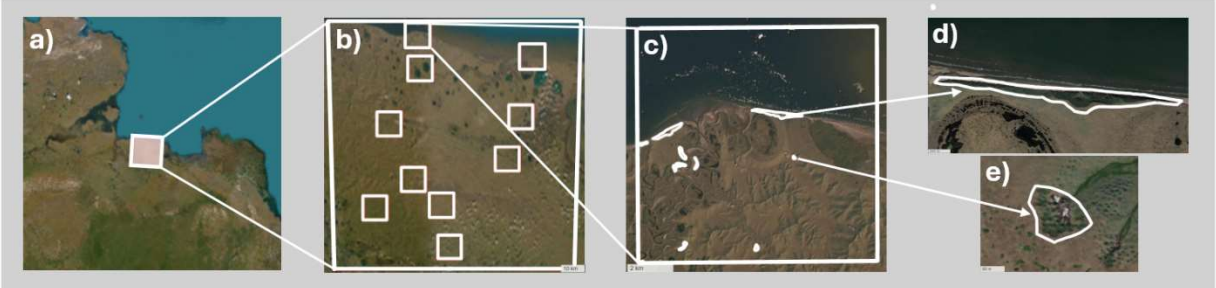


*Figure 4: **Validation dataset 2**. 5 main areas, such as **(a)** Northern Olenek, located in various positions across Siberia contain **(b)** 10 randomly located sub areas with **(c-e)** individual thaw slump features mapped out by Emma Schütt.*

1.2.2 Pan-Arctic Visualization of Landscape Change (2003-2022)

The Pan-Arctic Visualization of Landscape Change (2003-2022), also referred to as ALEX dataset, from Nitze et al. 2024b is based on Fraser et al. 2014; Nitze, Grosse 2016. The contact person is Ingmar Nitze (orcid.org/0000-0002-1165-6852). The extend of the data is pan-arctic, images of the satellites: Landsat-5 TM, Landsat-7 ETM+, and Landsat-8 OLI were used to detect landscape changes, the spatial resolution of one pixel is 30m x 30m, the temporal coverage spans from 2003-07-01 to 2022-08-31. The data shows spectral information on land surface changes in form of tasseled cap index derivatives of wetness (“TCW_slope”), greenness(“TCG_slope”) and brightness(“TCB_slope”). Each Pixel contains one value per tasseled cap index. The data is provided by the Alfred Wegener Institute Helmholtz Centre for Polar and Marine Research.

Each of the tasseled cap indices has an associated spectral slope, where 'slope' refers to a linear trend, fitted through the tasseled cap index value calculated on annually aggregated data (Nitze, Grosse 2016). The change of the spectral information recorded over 20 years of time is summarized in one value per property and pixel, for each of the raster cells. The three slope type values are represented in their corresponding colour shown in Figure 5. The colour shows if there was an increase or decrease of one or more slope types happening for each cell. Since there are several ways of how e.g. a trend of increasing brightness can appear in a landscape each of the colours that can emerge by mixing red, green and blue can be the result of several processes of the landscape. Importantly, the dataset only shows how the landscape changed not what the change was induced or controlled by.

Figure 5: ***Colourmap of spectral slope data.*** *The colour of each image pixel is determined by the relative strength of all three tasseled cap index slopes at this point.*

RTSs are slope failure features with the essential characteristic of containing a headwall (Nesterova et al. 2024). The headwall and parts of the slump floor, called scar zone, are transitioning from vegetation to dark wet surfaces. Those parts appear blue in the ALEX dataset, since the melting ice increases the wetness. The slump floor is often shown colourful compared to the surrounding landscape. The colour depends on the intensity of different change processes. Orange and yellow colours show stabilizing RTS parts, that change from muddy surface to vegetation. Figure 6 displays a selection of different RTSs and how they appear within the ALEX dataset. Notice that the area that got eroded during the observation period, including the headwall, is always shown in blue. The combination of having only one value per pixel that includes multiple types of information and the fact that RTSs should be clearly visible due to their nature of altering the surface makes the ALEX dataset perfect for looking at spectral variability and temporal dynamics of RTSs.

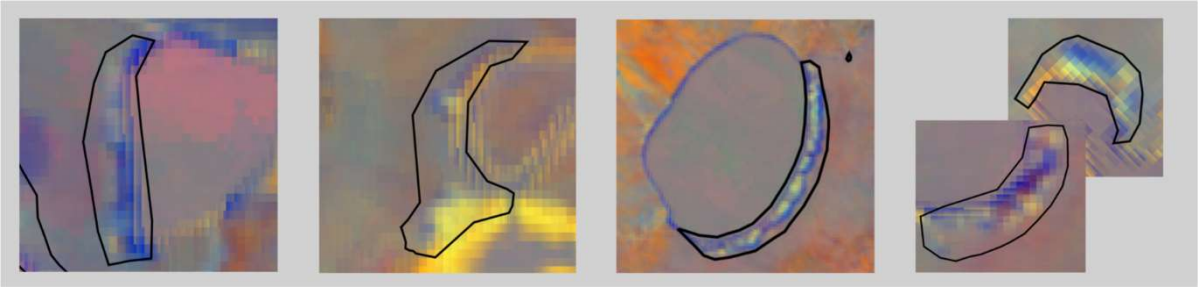


Figure 6: ***RTS appearance in the ALEX dataset.*** *Example RTS from the Chokurdakh region show how different RTSs can appear in terms of the spectral slope.*

1.2.3 Esri World Imagery Map

Esri World Imagery Map (Esri, Maxar, Earthstar Geographics, and the GIS User Community). The layer *World Imagery* was used, which includes one meter or better satellite and aerial imagery in many parts of the world and lower resolution satellite imagery worldwide. The single images are regularly updated (typically max. 3-5 years old). The data is provided and managed by Esri. A table including the citations of each image used can be found in the appendix (Appendix C1, Table 3).

1.2.4 JRC Global Surface Water Mapping Layers

The JRC Global Surface Water Mapping Layers, v1.4 (Pekel et al. 2016) dataset was generated using scenes from Landsat 5, 7, and 8 acquired between 16 March 1984 and 31 December 2021. The spatial resolution is 30 m. The dataset has seven different bands that include information on the presence of water: changes in occurrence, seasonality and persistence. The data is provided by EC JRC/Google. In this work the layers occurrence, seasonality, and change_abs were used.

1.2.5 Global Shoreline Dataset

The Global Shoreline Dataset (Sayre et al. 2019) was commissioned by the Group on Earth Observations. The product is a Global Shoreline Vector (GCV) with 30-m spatial resolution, developed from annual composites of 2014 Landsat satellite imagery. The GSV separates terrestrial (land) from marine environments (sea). Three separate layers indicate: continental mainland, islands greater than 1 km^2, and islands smaller than 1 km^2. The last update of the data set was on 2020-05-08. In this work only the mainland polygon layer was used.

1.2.6 Lake Dataset

The currently unpublished Lake Dataset from Ingmar Nitze (orcid.org/0000-0002-1165-6852) is a product with a spatial resolution of 30 meters. It is created by using satellite data of Landsat 5, 7 and 8 and the methodology of Nitze et al. 2017; Nitze et al. 2018. The temporal coverage includes the years 2000 to 2020. A pan-arctic extend is planned for the dataset. The actual version comprises 20 different attributes of which geometry is the one that is important for this work.

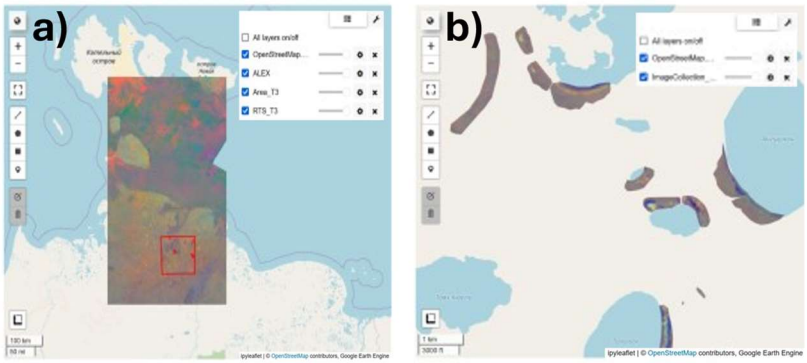
2 Methods

The analysis was performed in a Google Colaboratory Notebook (03.11.2024 - 12.01.2025 Version 1.1.0 (Colaboratory-team 2023); 13.01.2025 – 15.03.2025 Version 1.2.0 (Colaboratory-team 2024)) and in a QGIS 3.34 Prizren (QGIS Development Team 2023) environment. The Google Colaboratory Notebook code is attached as an appendix (Appendix D). Several toolboxes were used, the “ee” and the “geemap” (Wu 2020) toolbox were part of the key tools. Details of the analysis are presented with respect to the (sub-) hypothesis that they relate to.

2.1 Comparison of cumulative RTS spectral slopes between study areas

To compare the cumulative spectral slope between different areas (sub-hypothesis 1), histograms that contain all RTS data for each region were calculated. First, the ALEX dataset (section 1.2.2) was loaded as image collection and the Validation Dataset 2 (section 1.2.1) data was loaded as feature collections containing RTS outline geometries. The data extraction and preparation workflow are summarized in Figure 7. The rescaling step in Figure 7d was necessary, because the data was stored as 8 bit integers (numbers from 0 to 255) and had to be restored to the original range of -0.12 to 0.12. It is important to use the correct values for the plots to show more intuitively that half of the range shows a decrease (-0.12 to 0) and half of it shows an increase (0 to 0.12) of the spectral index in the observed time. Considering the spatial resolution (30 m x 30 m per pixel) of the data, the histogram frequency axis can be rescaled to area (m^2) by multiplication with 900.

To view the data within their geographical context and to compare differences in spectral slope between the areas the histograms were visualized on a map (Figure 11 in results). The map is projected using the Russia Polar Stereographic coordinate system (EPSG: 5940) and overlaid with a WGS 84 graticule for orientation. Additionally, a violine plot of the same data (Figure 12) shows the five distributions for a simplified comparison.



c)

	TCB_slope	TCG_slope	TCW_slope
0	135	122	119
1	137	124	118
2	136	126	120

d)

	TCB_slope	TCG_slope	TCW_slope
0	0.007059	-0.005176	-0.008000
1	0.008941	-0.003294	-0.008941
2	0.008000	-0.001412	-0.007059

e)

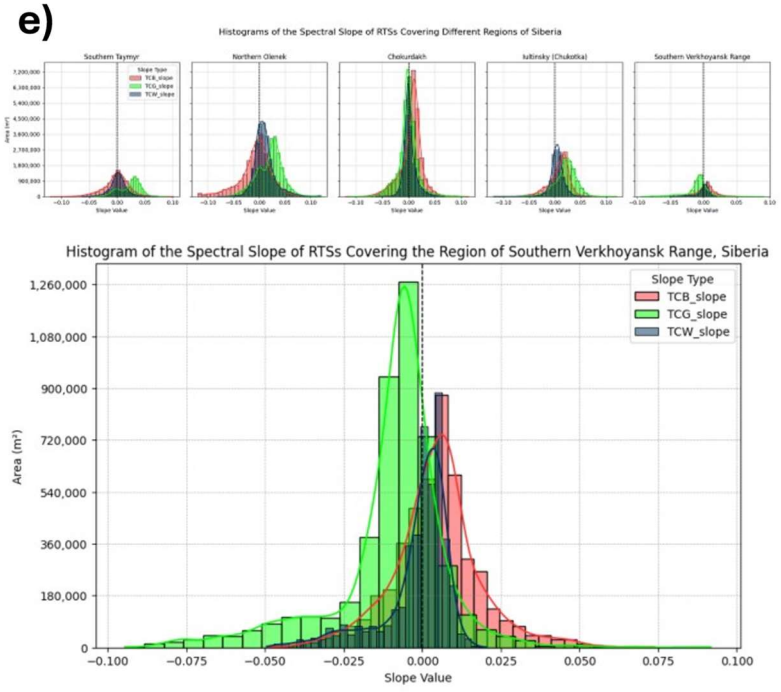


Figure 7: ***Workflow for calculating histograms that contain all RTS data for each region. (a)*** *Selection of relevant Alex images per region.* ***(b)*** *Creation of one ImageCollection per area with ALEX data that is clipped to the polygon geometry of the RTSs.* ***(c)*** *Creation of one GeoDataframe (gdf) per area. (Only an excerpt of the data frame is shown.)* ***(d)*** *Rescaling of the data to the range in which it was originally measured. (Only an excerpt of the data frame is shown.)* ***(e)*** *Calculation of cumulative spectral slope histogram per area. (The small histograms contain the same axis labels as the larger one.)*

In a second step, the significance of differences in spectral slope was validated using statistical analysis. The normality of all groups (i.e., one group constitutes of all values of one spectral slope type from one area) was assessed using both, the Shapiro-Wilk test and the Anderson-Darling test. These tests were chosen because they differ in sensitivity

to deviations from normality: the Shapiro-Wilk test is more powerful, while the Anderson-Darling test pays more attention to the tails of the distribution (Mohd Razali, Nornadiah and Yap, Bee 2011). Since normality was not necessarily given, the Kruskal-Wallis test was used as a non-parametric alternative to ANOVA to evaluate whether the groups originated from the same distribution. This test determines differences in the distributions of the groups by comparing their ranked values rather than their means, with the null hypothesis stating that all groups follow the same distribution (McDonald 2014). Finally, a post-hoc Dunn-Bonferroni test was performed to conduct multiple pairwise comparisons and identify which groups significantly differed from each other, while controlling for the family-wise error rate. This adjustment is necessary to reduce the likelihood of Type I errors that arise when conducting multiple statistical tests. (McDonald 2014). The null hypothesis for the Dunn-Bonferroni test states that there is no difference between the compared groups. Only the results of the last statistical test (Dunn-Bonferroni) are shown in section 3.1, results, visualized as heat map (Figure 13), since the results of the statistical tests are based on each other.

2.2 Spectral slope comparison within study areas

To analyse whether the sub areas within one main area show significant differences regarding spectral slope (sub-hypothesis 2), another statistical analysis was performed. The analysis was conducted two times: one time with all values that are included in each RTS geometry of each sub area. Another time where each RTS geometry had a 5 m inward buffer applied, to cancel out the 5 m buffer that was added to each RTS geometry during the data collection (see section 1.2.1). This was done to reduce the signal from values in the RTS polygon buffer areas, which aren't part of the RTS themselves.

The statistical test routine was the same as the one for sub-hypothesis 1 (section 2.1). With the exception that only the more powerful Shapiro-Wilk-test was used to test for normal distribution and that, in this case, a group consists of all values of one spectral slope type from one sub area.

The significant differences between the groups were classified using the classification of similarity shown in Figure 8. The classes span from highly heterogeneous (the sub area is dissimilar to all other sub areas) to highly homogeneous (the sub area is similar to all

other sub areas). In between, there are the classes heterogeneous and homogeneous (the sub area is dissimilar resp. similar to most other sub areas). The classification of the single sub areas was done according to the scheme shown in Figure 9.

The resulting pie charts were visualized in an overview map to place them into geographical context. The map (Figure 14) is shown in the Russia Polar Stereographic coordinate system (EPSG: 5940) projection and is overlaid with a WGS 84 graticule for orientation.

Definition of similarity classes:

Similarity classes describe the proportion of similar vs. dissimilar central tendencies of the spectral slope of sub areas (sub areas within the same main area were pairwise compared).

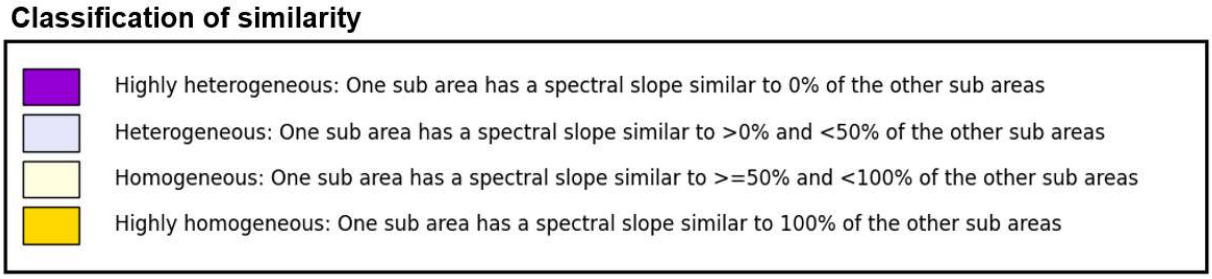


Figure 8: ***Classification scheme of spectral slope similarity.*** *Applied after carrying out pairwise comparisons using the Dunns-test to simplify the interpretation of the results.*

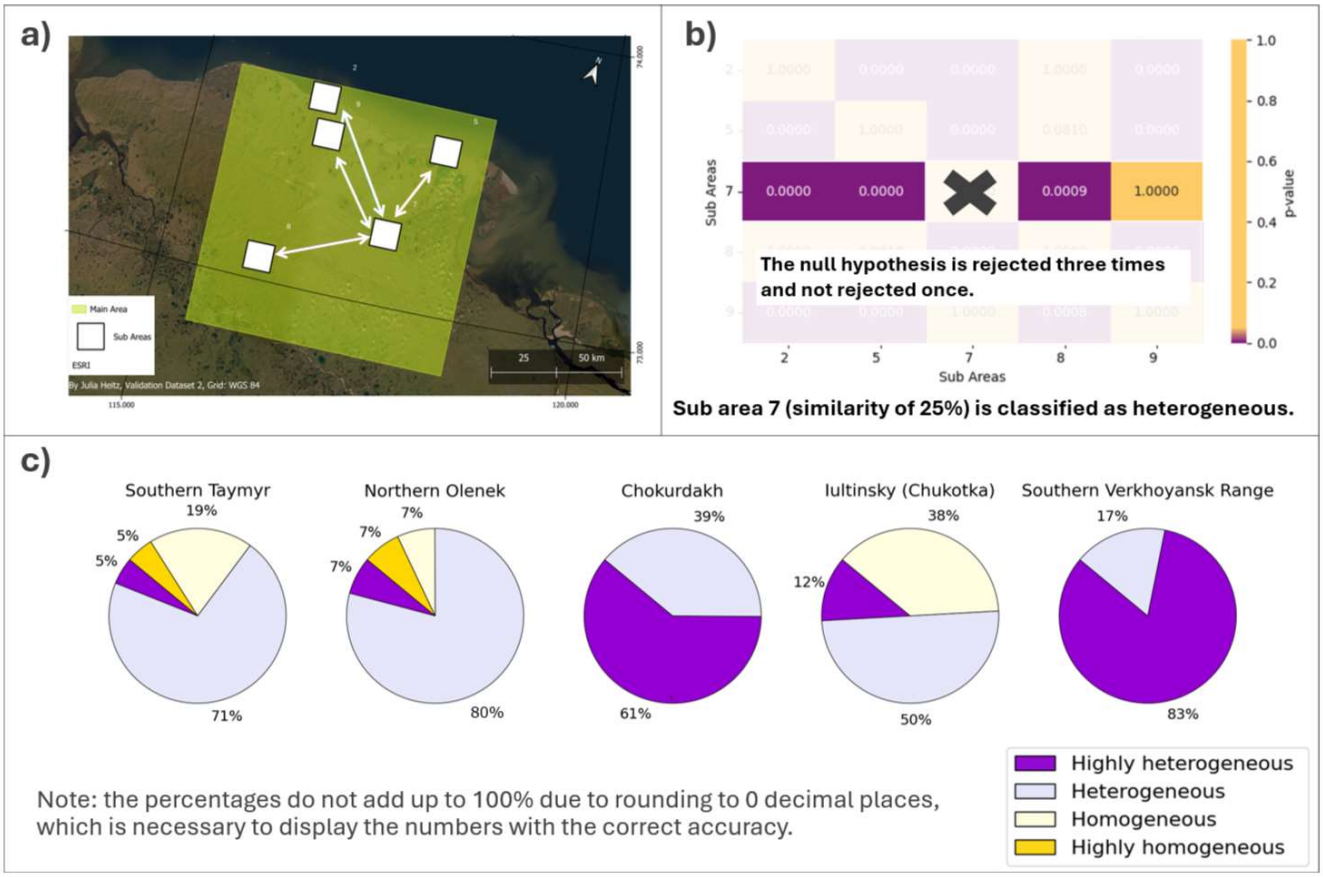


Figure 9: ***Workflow for similarity classification. (a)*** *Comparing one sub area (e.g. sub area 7) with all other sub areas (e.g. sub areas 8,9,2,5) within the same main area (e.g. Northern Olenek) using statistical tests.* ***(b)*** *The results of the pairwise comparison are recorded in a matrix, which is investigated row by row (e.g. row 3 corresponding to sub-area 7). The fraction of rejected null hypothesis (the compared sub-areas show a significant similarity), ignoring the self-comparison, is expressed as percentage and classified using the classification scheme of Figure 8.* ***(c)*** *The classification summary is displayed as one pie chart per main area.*

2.3 Terrain position and spectral slope similarity

To conduct an analysis regarding similarities of spectral slope of those RTSs which are both within the same main area and located at the same terrain position (sub-hypothesis 3), a new dictionary is needed. This dictionary should include data frames containing the spectral slope data per RTS and link these to the terrain position (TP) of each RTS. To create this dictionary, the spectral slope data from the geodata frames of sub-hypothesis 1 was divided into individual RTS data frames and supplemented with the terrain position of each RTS, as well as the RTS geometry. Both terrain positions and RTS geometry were taken from other datasets, as outlined below.

Within this work, terrain positions were sorted into the categories lake shore, sea shore, river shore and gully. Out of necessity, an additional TP class for the combination of ponds+gullies was introduced in the course of the analysis. Classification was performed by comparison with the mainland polygons layer from the Global Shoreline Dataset (section 1.2.5) and the Lake Dataset (section 1.2.6).

Sea shore category RTSs were found by overlaying the mainland polygon with the RTS outline polygons and identifying those RTS polygons that are not fully contained in any of the mainland polygons. Since the purpose of the mainland polygon is to separate land and sea, polygons that include more than land need to be located at the sea shore. Figure 10a is a good example of a sea shore (TP class 1) RTS.

Lake shore RTSs (TP class 2) were identified by looking for intersections in between RTS geometries and the geometry of lakes. Figure 10b is an excellent example for a lake shore RTS.

The classes river (TP class 3) and gully (TP class 4) had to be assigned by hand because of a lack of available Siberian river and gully datasets. The intermediate result dataframes were imported into QGIS, where each un-categorized RTS was investigated individually. Reference layers that aided the TP class assignment were Esri World Imagery Map (1.2.3) and the layers *occurrence*, *seasonality* and *change_abs* from the JRC Global Surface Water Mapping Layers (1.2.4). Esri world imagery helped to identify evident features such as gullies or lakes.

The JRC Global Surface Water Mapping Layers aided with decisions involving unclear waterbody shores and to assess the waterbody impact in general. Waterbody impact may have occurred in the past if an RTS used to be located at the shore of a currently drained lake. Indicators for waterbody impact are if the RTS is close to where water frequently occurred in the time from 1984 to 2021 (JRC *occurrence* layer, scale 0 to 100%), and if the layer *change_abs* indicates a strong absolute change in water occurrence in between two timespans (1984-1999 vs 2000-2021, scale -100% to 100%) close to the RTS position. Both options indicate that the RTS at said drained lake shore was strongly impacted by the lake shore in the past and therefore a lake shore (TP class 2) assignment would be appropriate. An image of such a TP class 2 assignment is presented in Figure 10c.

The guidelines for the individual class assignments were:

- RTSs that predominantly overlap with one terrain position feature are assigned to that TP class
- RTSs that predominantly follow the outlines of a terrain position feature are assigned to that TP class.

In the course of analysis, the classification scheme had to be expanded to include all striking TP features. Several RTSs were observed close to ponds which were connected to gullies. Since these RTS could be classified neither as lake nor as gully, a new TP class 6: "pond + gully" was created (see example RTS in Figure 10d). All TP classes are shown in Table 1. Fully categorized datasets were returned to the Google Colaboratory Notebook for further analysis.

Table 1: Classes of terrain positions

Terrain Position Name	Terrain Position Class
Sea	1
Lake	2
River	3
Gully	4
(Others)	(5)
Ponds + Gully	6

The geometries of the RTS polygons in the TP data frames were used to divide the spectral slope data of the buffer corrected geodata frames of sub-hypothesis 2 (section 2.2) into new RTS specific data frames. The dataset included the TP class and the main area from which the RTS originated. This dataset enabled the analysis of all RTS belonging to each TP class, for each main area. The analysis was performed similar to that of sub-hypothesis 1 and 2. Different from the analysis of sub-hypothesis 1, only the more powerful Shapiro-Wilk-test was used to test for normal distribution and, this time, a group consisted of all values of one spectral slope type from one RTS. The results were p-value matrixes that show if RTSs within the same main area and the same TP class show significant similarities. The matrixes were classified according to their similarity with the classification workflow shown in Figure 9, with the difference, that this time the classification applied to RTSs instead of sub areas.

The results are hierarchical, with more layers of complexity than a normal pie chart can visualise. An alternative is the multilayered plot called 'sunburst chart'. Starting from the broadest category, the first layer of the plot represents a standard pie chart. The next layer of detail represents the contents of the first layer pie slices, and so forth. The python package *plotly.express* (Kruchten, N., Seier, A., Parmer, C. 2024) was used for sunburst charts in this work. It is important to note that certain constraints apply to data accepted

by sunburst plots. For example, it is mandatory to convert the percentages of the similarity classes to proportional count values such that every pie piece has the right dimension. The numbers visible in the charts (for example of Figure 17) are nevertheless the original percentages. This conversion inherently suffers from a rounding uncertainty, since the percentages are rounded to 0 decimal places in the first step, and the count numbers derived from the percentages are rounded again. The uncertainty varies for each pie piece of the plot, depending on the corresponding count value. The formulas and error tables for the conversion of percentages to count can be found in the appendix (Appendix C5).

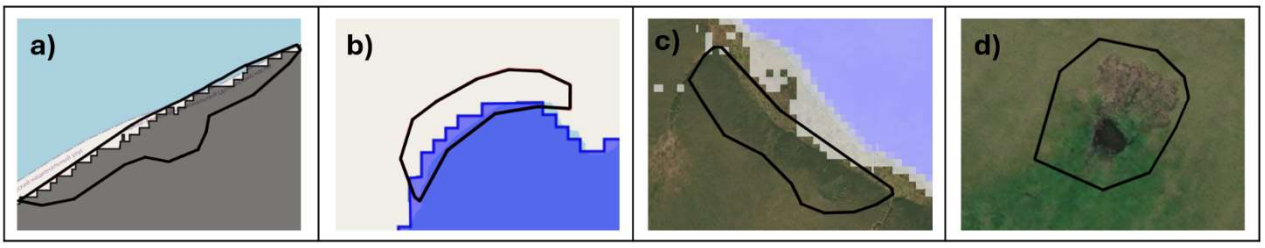


Figure 10: ***Examples for TP classification****. The black outlines are the geometry of the RTSs, the background colours correspond to the terrain type.* ***(a)*** *Example sea shoreline where the RTS polygon is not fully covered by the mainland polygon, resulting in TP class 1 assignment.* ***(b)*** *Example lake shoreline where the RTS polygon and the lake polygon are overlapping. This RTS is assigned to TP class 2.* ***(c)*** *Manual assignment example 1: this RTS is located far from the current water level of the lake, however, both the* occurrence *and* change_abs *layers of the JRC Global Surface Water Mapping Layers show the proximity of the water body in the past. The previous proximity to the lake implies a strong impact of the lake in the RTS development. This RTS is assigned to TP class 2.* ***(d)*** *Manual assignment example 2: this RTS is located at a small pond, which is connected to a gully. Since the RTS is neither located at a lake nor directly at a gully the TP class 6 'pond + gully' was created. This RTS is assigned to TP class 6.*

2.4 Morphology and spectral slope similarity

To investigate the similarities of RTS spectral slopes within the same main area and containing the same RTS morphology (sub-hypothesis 4), the dictionary used for the terrain position analysis above was expanded. In addition to the TP class and the RTS geometry, information on the morphology was needed. Consequentially, each slump had their morphology class assigned manually in QGIS. Possible classes are: thermocirque, thermoterrace and the combination of both, as described in (Nesterova et al. 2024), see Figure 1. Layers that were used for the assessment were the basemap: Esri World Imagery Map (1.2.3) and the geometry of the RTS layers (1.2.1) themselves. The extended data frames were imported back to the Google Colaboratory Notebook. The geometry of the

RTS polygons in the morphology/TP data frames were used to divide the spectral slope data of the geodata frames of sub-hypothesis 2 (the ones that include the 5 m inward buffer) into new RTS specific data frames. The morphology class, the TP class and the main area from which the RTS originates were included in the RTS data frame names. From this point forward, the analysis followed that of sub-hypothesis 3, with the difference that morphology was considered instead of the terrain position. Again, the statistical analysis was followed by similarity classification and, finally, the data was presented in sunburst charts. The formulas and error tables for the conversion of percentages to count during the sunburst chart visualization can be found in the appendix (Appendix C5). The manually created terrain position and morphology data frames are stored at the Alfred Wegener Institute, section Permafrost, group Permafrost Remote Sensing, and are available upon request.

3 Results

To aid readability, the sub-chapters of the results section follow the order of the sub-hypothesis, similar to the methods section above. Therefore, the results presented in section 3.1 are the result of the methods described in section 2.1, and so on.

3.1 Comparison of cumulative RTS spectral slope between study areas

To compare the spectral properties and spatial relationships of the five study areas, the cumulative spectral slope histograms are plotted on a map of Siberia in Figure 11. From visual inspection, the distributions of the slope types shown in the histograms vary between the study areas regardless of the distance of the areas towards each other. The peak of the greenness distribution of the area Southern Verkhoyansk Range (outlined on the map with an orange frame) shows the highest spectral slope decrease. For the area Chokurdakh (outlined on map with a purple frame) all three distributions are quite centred on top of each other but differ in height. The other three areas: Southern Taymyr (outlined on the map with a blue frame), Northern Olenek (turquoise frame) and Iultinsky (pink frame) all show differences in the height of the distribution peaks and its centres.

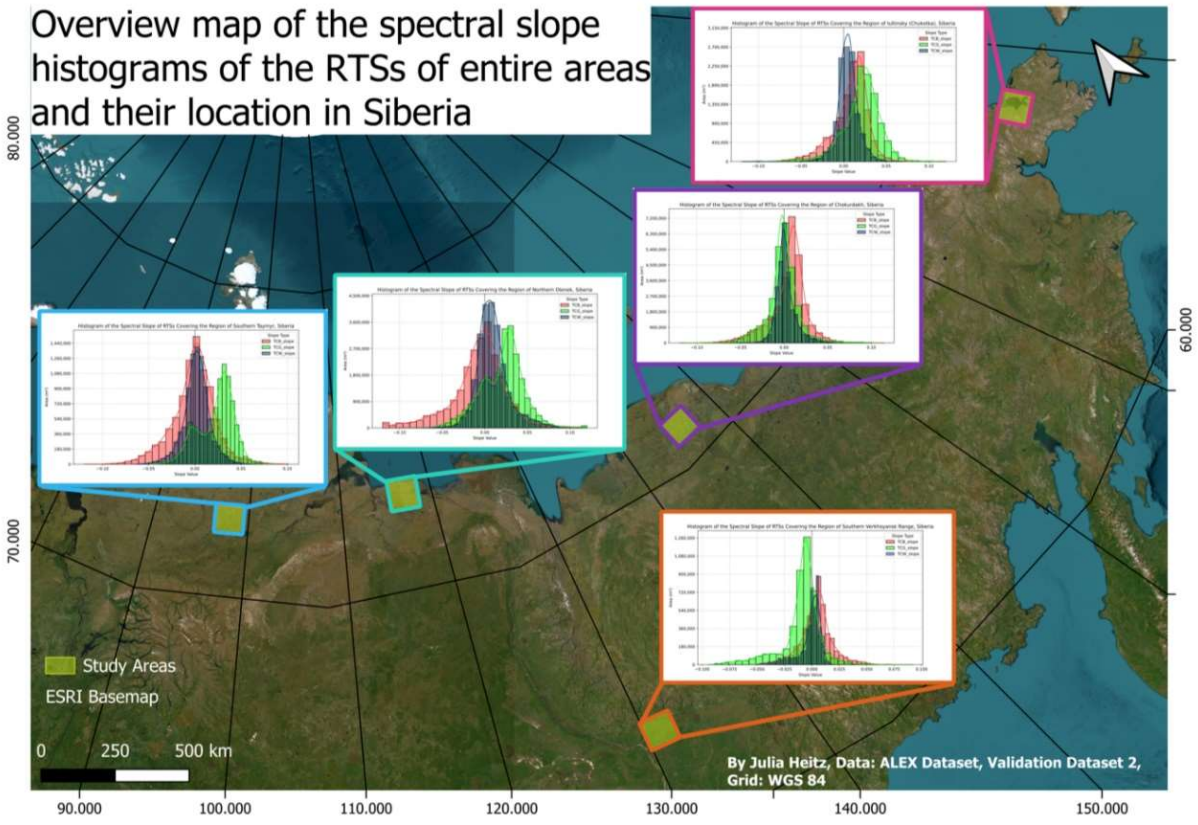


Figure 11: ***Cumulative spectral slope and spatial relationship of all study areas****. The outline colour of each histogram corresponds to the outline colour of the corresponding area.*

To ease the comparison of the individual distributions, they are presented in Figure 12 as a violine plot. Evidently, no two spectra are exactly alike. However, there are some features that appear in multiple spectra.

There is some consistency in that only slope type greenness ('TCG_slope') shows a double peak. The double peak feature is observed strongly in three out of five distributions (the greenness for the areas Southern Taymyr, Northern Olenek and Iultinsky), while the other two have elongated tails. The greenness ('TCG_slope') distributions for Chokurdakh and Southern Verkhoyansk Range have a mean value below zero, indicating that there is a net reduction of greenness in these regions in the observed timeframe.

The distributions of brightness ('TCB_slope') tend to be flatter and longer than those of wetness ('TCW_slope'). The distributions of Southern Verkhoyansk Range for brightness and wetness are noticeably shorter than those of the other areas. The area of Southern Verkhoyansk Range that is affected by RTSs is very small. The elongated distribution of Southern Verkhoyansk Range greenness must have strong outliers in the limited sample set. In general, wetness distributions tend to be narrower.

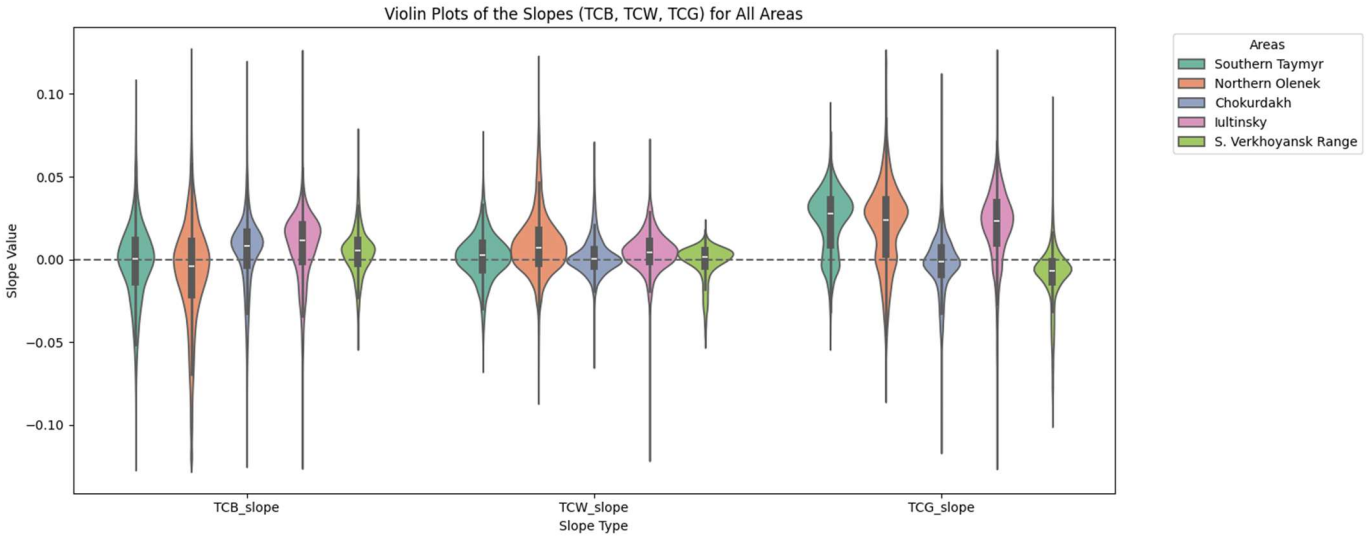


Figure 12: ***Violine plot showing the spectral slope types of each study area.*** *The x-axis shows the different spectral slope types, the abbreviations TCB, TCW and TCG stand for the term tasseled cap index of brightness, wetness or greenness. The distributions shown in the plot are not normalized and are based on different sample sizes.*

The results of a Dunn-Bonferroni pairwise area comparison is presented in Figure 13. The null hypothesis of the Dunn-Bonferroni test is that the compared areas are similar to each other. Yellow fields of Figure 13 correspond to p-values that exceed the significance level

on 0.05 (don't reject the null hypothesis). Since none of the off-diagonal terms exceed the significance level, the alternate hypothesis must be accepted: there is a difference between all the compared areas. Consequentially, the spectral slopes between the main areas are significantly different and sub-hypothesis 1 cannot be rejected.

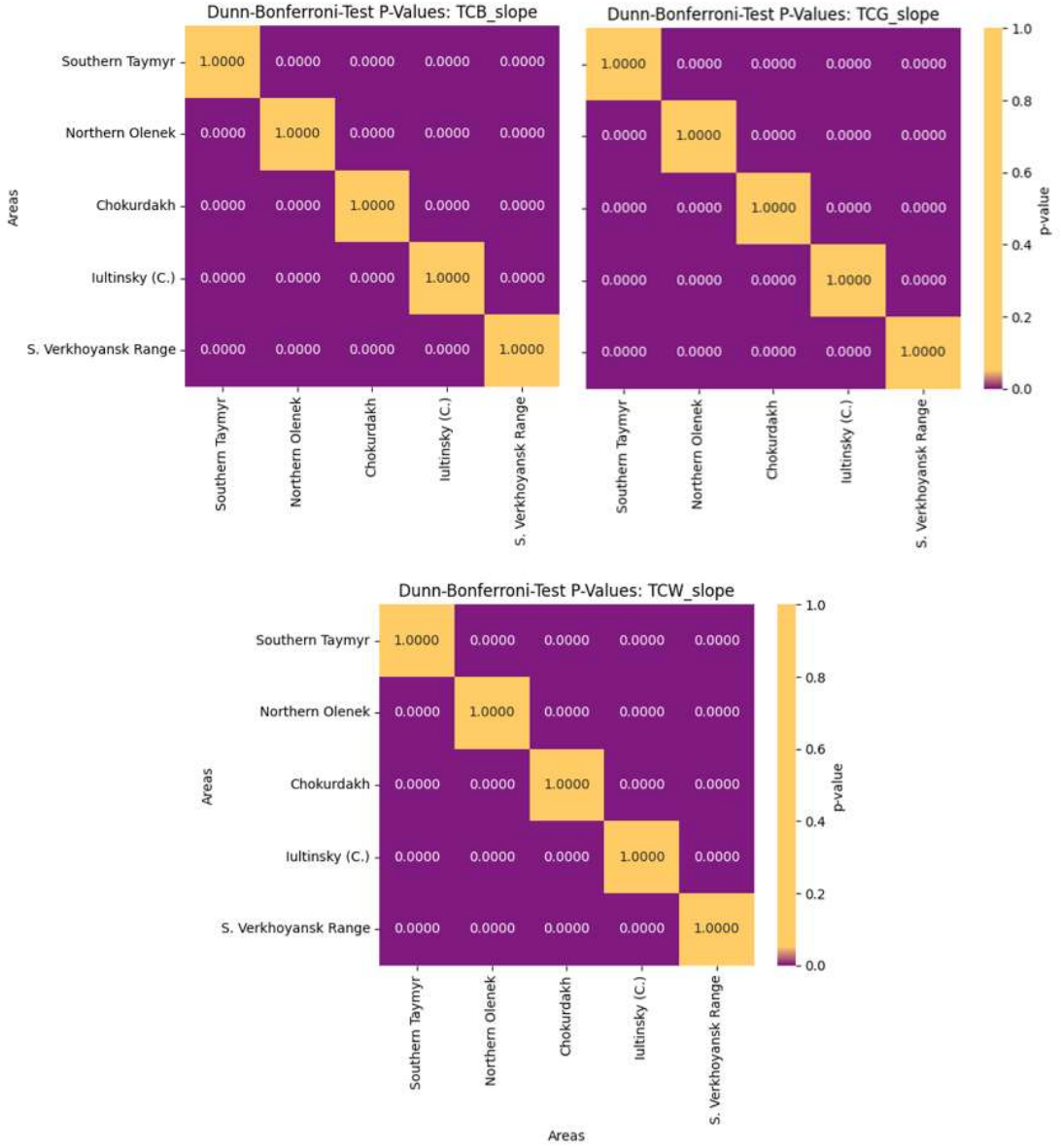


Figure 13: ***Heatmap of the Results of the Dunn-Bonferroni test.*** *P-values that exceed the significant level of 0.05 are shown in yellow. None of the off-diagonal terms exceed the significance threshold. The diagonal terms are self-comparisons of areas that don't contribute valuable information on similarity.*

3.2 Spectral slope comparison within study areas

The process described in section 2.2 leads to a classification of the similarity of spectral slopes, classified by the scale presented in Figure 8. A convenient visualisation of the fraction of similar/dissimilar spectral slopes is to plot a pie chart of the similarity classes. Figure 14 shows an overview of the spatial relationships between such pie charts of the

distribution of sub area similarity classes for each main area.

If sub-hypothesis 2, all sub areas are dissimilar to all other sub areas, would be completely true, all pie charts would be coloured dark purple and classified as highly heterogeneous. Every pie chart contains a fraction of sub areas that are classified as highly heterogeneous but, at the same time, each pie chart also contains at least one other class. Therefore, sub hypothesis 2 can be neither rejected nor accepted. This is why sub-hypothesis 2 is only partially rejected. The spectral slopes of the sub areas (within a region) are mostly different, the classes highly heterogeneous and heterogeneous combined always make up more than 50% of the pie, but there is always some similarity between some of the sub areas. The amount of similarity varies a lot. Northern Olenek and southern Taymyr include sub areas which are both homogeneous and highly homogeneous. Iultinsky (Chukotka) includes the largest percentage of sub areas that were classified as homogeneous. And Chokurdakh and Southern Verkhoyansk Range only include sub areas of the similarity classes heterogeneous (a sub area is dissimilar to most other sub areas) and highly heterogeneous.

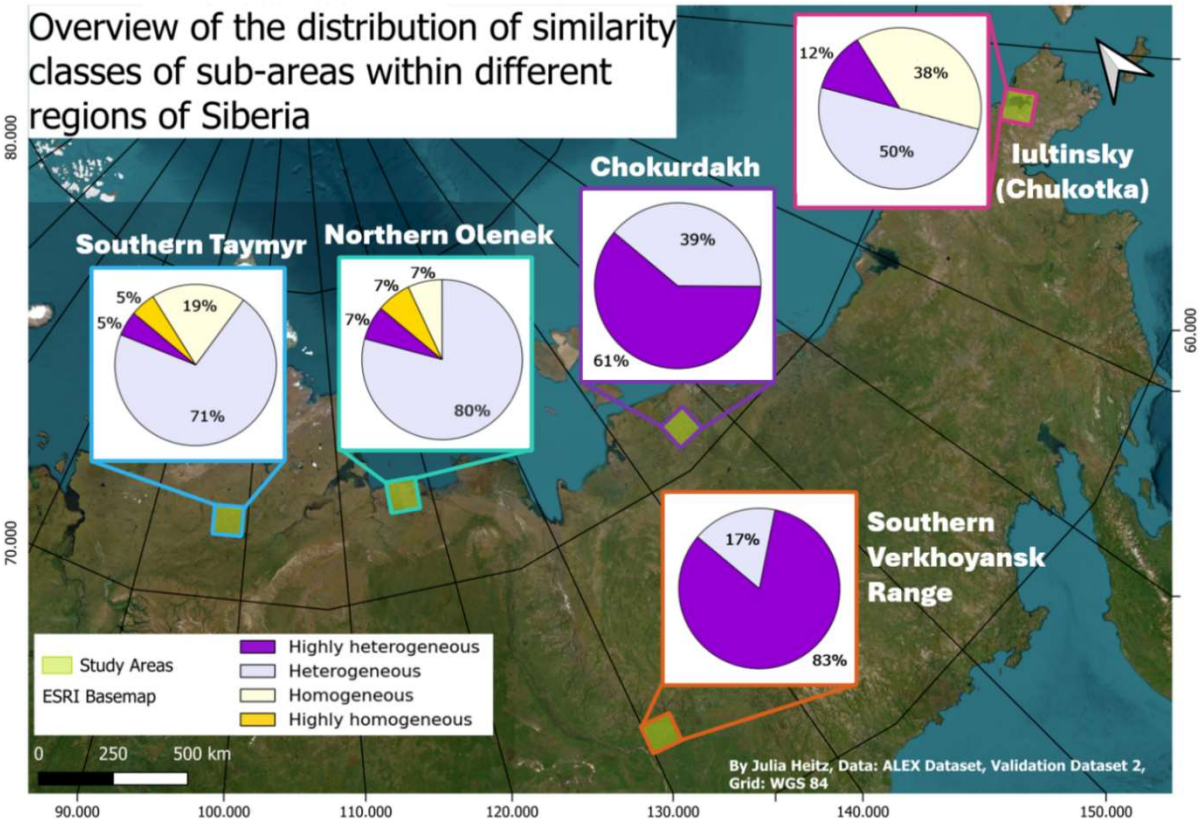


Figure 14 ***Similarity class pie charts vs. study area location.*** *The outline colour of each pie chart corresponds to the outline colour of the area which is shown the pie chart.*

One aspect to consider in a similarity inspection is that the RTS geometries were recorded with a 5 m buffer. Including non-RTS areas may influence the similarity analysis. Figure 15 shows the results of an identical analysis with an additional 5 m inward buffer to compensate the original buffer. Slight changes are observed for two of the five areas, compared to Figure 14. For Iultinsky the percentage of highly heterogeneous decreased by 4% while the percentage of heterogeneous increased by 13%. The percentage of homogeneous decreased by 9%. In Southern Taymyr the percentages of highly homogeneous and of highly heterogeneous stayed the same, while heterogeneous decreased by 10% and homogeneous increased by 10%. A more detailed analysis of the impact of the 5 m buffer on the RTS area can be found in the Appendix C3.

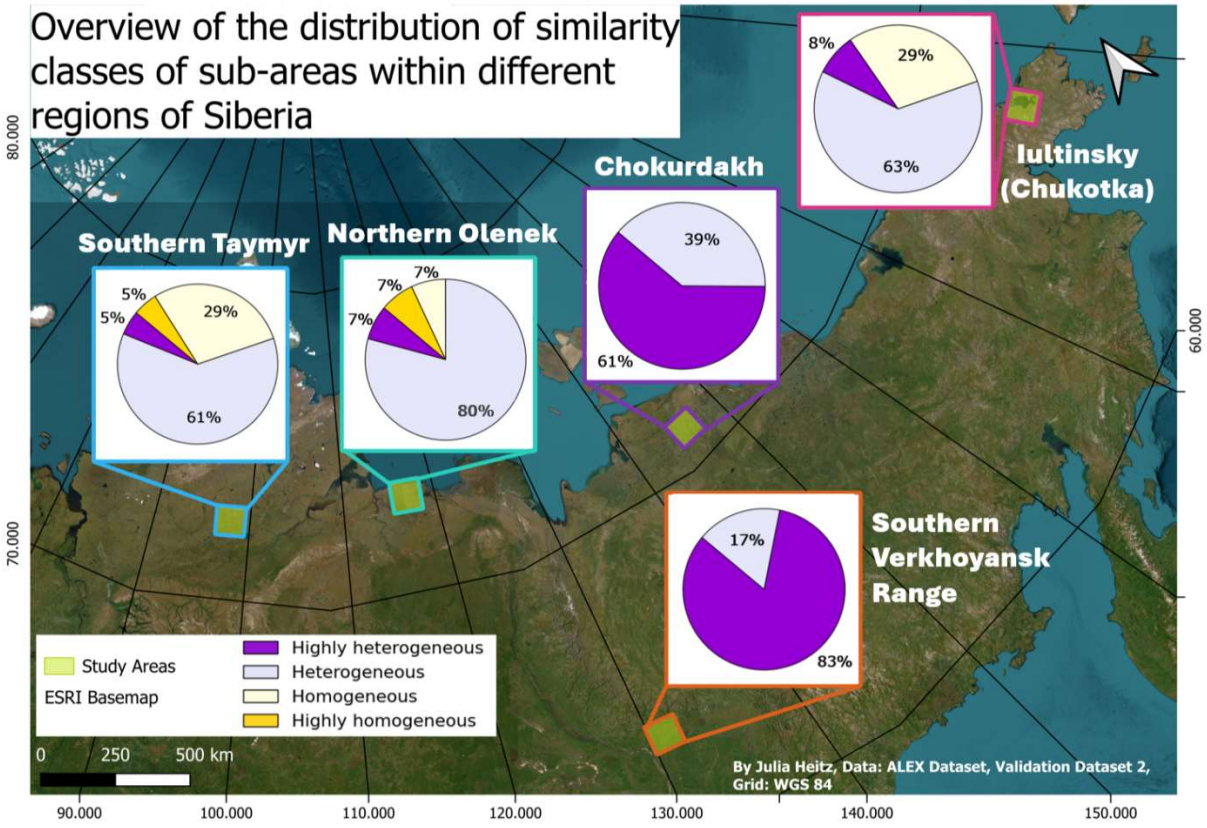


Figure 15 ***Similarity class pie charts with 5 m inward buffer vs. study area location.*** *The outline colour of each pie chart corresponds to the outline colour of the area which is shown the pie chart. Note that the sum of the percentages may exceed 100% due to rounding errors.*

3.3 Terrain position and spectral slope similarity

The creation of the new data frames that contain the geometry and the terrain position of each RTS (see section 2.3) made it possible to investigate the fractions of RTSs located close to the terrain positions of the study areas.

The pie charts of Figure 16 show that the majority for the RTSs in Southern Taymyr (98%), Northern Olenek (70%) and Chokurdakh (70%) belong to the terrain position class lake shore. In Iultinsky lake shores are the TP for 46% of the RTS which is the largest pie piece

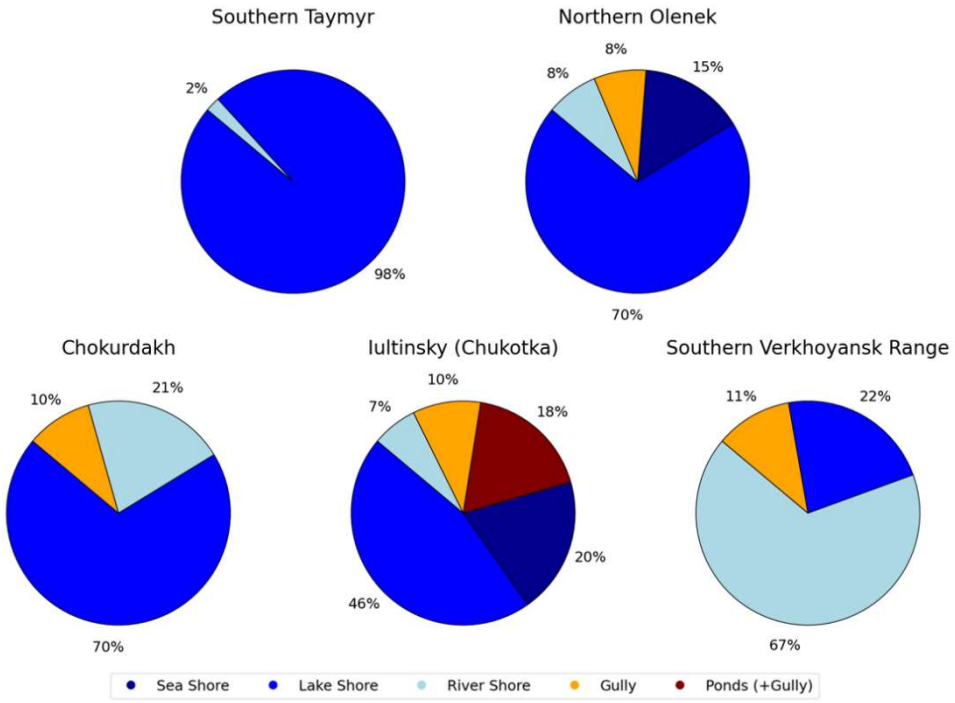


Figure 16: ***Pie charts of the terrain position percentage for each area.*** *The colour of the pie pieces corresponds to the terrain position.*

for that area. Southern Verkhoyansk Range is the only area where lake shores don't make up the largest pie piece, taking second place with 22%. This shows that lake shores are, in general, the dominant TP.

River shore is a terrain position that is also included in all five study areas. Southern Taymyr has the smallest portion with 2%, Iultinsky the second smallest with 7%, followed by Northern Olenek with 8%. 21% of the RTSs in Chokurdakh are located at river shores and in Southern Verkhoyansk Range river shores are the majority the terrain position with 67%. Gullies were found as terrain positions for the areas: Northern Olenek, Chokurdakh, Iultinsky and Southern Verkhoyansk Range, with fractions between 8% and 11%. Iultinsky was the only study area in which RTSs with the TP Pond + Gully were found.

The sub-hypothesis 3 states that: the spectral slope of single RTSs that are located at a certain terrain position are always the same as the spectral slope of single RTSs that are located at the same terrain position within the same main area. If this would be completely true, all of the RTS should be classified as highly homogeneous (yellow),

indicating that the RTS is similar to all other RTS located at the same terrain position (a modification of the classification scheme in Figure 8).

Sunburst charts, such as Figure 17 to Figure 21 below, are visualisations of hierarchical data. The central circle represents 100% of the RTSs of one area and is labelled with the study area name and the total number of RTSs of that area. The next ring is divided into segments corresponding to the fraction of the RTSs contained in each TP class, similar to a standard pie chart. The segments are labelled with the name of their TP class and the number of RTSs which are associated with them. The outermost ring shows the distribution of similarity classes within each TP class. The colour scheme follows a hierarchical blending approach. The outermost ring is coloured in colours that correspond to the similarity classes (see legend of plots). The colours of the second ring (the ones of the TP class) are the proportionally mixed colours of the outermost ring (based on the size of the similarity classes). The colour of the central circle results from a proportional blend of the TP class colours.

Figure 17 is a sunburst chart of similarity classes corresponding to RTS terrain positions for the study area Northern Olenek. The blending approach of the colour code reveals at first glance that Northern Olenek does not show much similarity between RTSs within the same TP. The single TP classes are either coloured grey (neutral) or in light blue (heterogenous range) colours. The central circle combines the similarity information in the weighted colour mixing scheme and indicates that the RTSs at the same TP within Northern Olenek are mostly heterogeneous. A closer look shows that the TP classes lake, river and sea are mostly heterogeneous. Only the TP class gully is homogeneous as largest similarity class (42%) but the classes heterogeneous (33%) and highly heterogeneous (25%) combined shift the TP class back into the heterogeneous part of the scale. This can be seen from the blue colour of the gully segment.

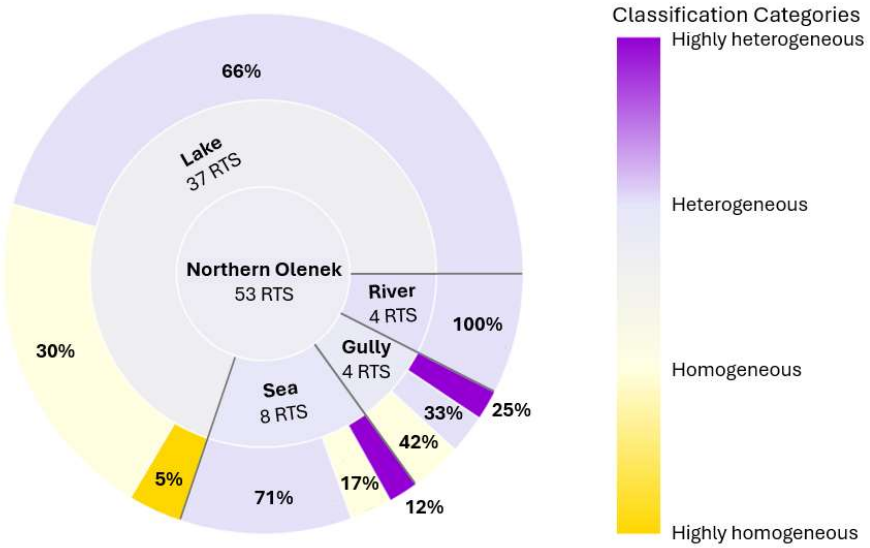


Figure 17: ***Sunburst chart of Northern Olenek, showing the similarity classes corresponding to the terrain positions.*** *The outermost ring shows the similarity class proportions of each TP. The second ring shows the fraction of RTSs that belong to each TP, the number of RTSs of that fraction is displayed. The central circle contains the total number of RTSs that are found in this area. The colour scheme follows a hierarchical blending approach. The initial colours are those of the similarity classes shown in the outermost ring. The meaning of the colours is shown in the legend. The TPs lake, river and sea are mostly heterogeneous (66%, 100%, 71%). The TP gully has a dominant similarity class of 42% homogenous which is overshadowed by the combination of 33% heterogeneous and 25% highly heterogeneous. In total, the RTS with similar TP in Northern Olenek are slightly heterogeneous regarding their spectral slope similarity.*

The sunburst chart of similarity classes corresponding to terrain positions for the study area Southern Taymyr is illustrated in Figure 18. The blending approach of the colour code in the central circle reveals that Southern Taymyr leans slightly towards homogeneous in total. However, Southern Taymyr has only one TP class that contains more than one RTS, which is a requirement for the similarity classification. This TP class is lake shore and is coloured in light yellow (slightly homogeneous). The TP class river contains just a single RTS and is left unclassified as a result. A detailed examination of the TP class lake shows that the similarity class highly homogenous (1%) and homogeneous (56%) are weighed up against 43% heterogeneous, underlining that the lake shore RTSs of Southern Taymyr are slightly homogeneous in total.

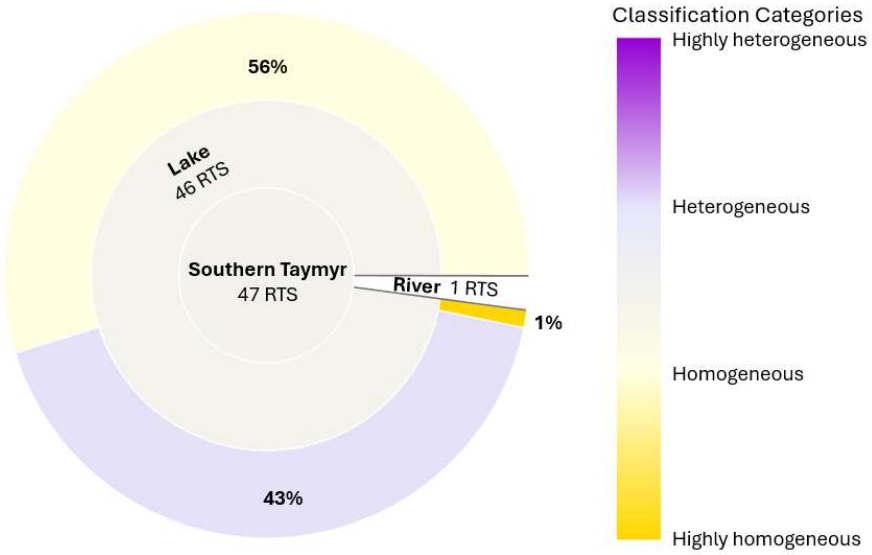


Figure 18: ***Sunburst chart of Southern Taymyr, showing the similarity classes corresponding to the terrain positions.*** *The majority of RTSs located at Southern Taymyr's only TP class* lake *are in the similarity class homogeneous (56%). The TP class river contains only a single RTS and is therefore excluded from the similarity classification.*

In Figure 19, the sunburst chart of similarity classes corresponding to terrain positions for study area Southern Verkhoyansk Range is displayed. The first important difference, compared to the other study areas, is that this area contains only 9 RTSs. The blending approach of the colour code shows at first glance that Southern Verkhoyansk Range shows very little similarity between RTSs within the same TP. The single TP classes are either coloured in blue (heterogeneous range) or in purple (highly heterogeneous range) colours. The central circle combines the similarity information for the whole area, it is also coloured light purple, which shows that the entire area is highly heterogeneous and nearly no similarity between the RTSs located at the same TP was found. Examining the TP classes in greater detail reveals that lake includes only two RTSs which have no similarity to each other in any of the spectral slope types, which results in the classification of 100% highly heterogeneous. The TP class river includes 6 RTSs, this TP class is dominated by the similarity class highly heterogeneous (39%) but the inclusion of 33% homogeneous and 28% heterogeneous keeps the river segment from turning purple.

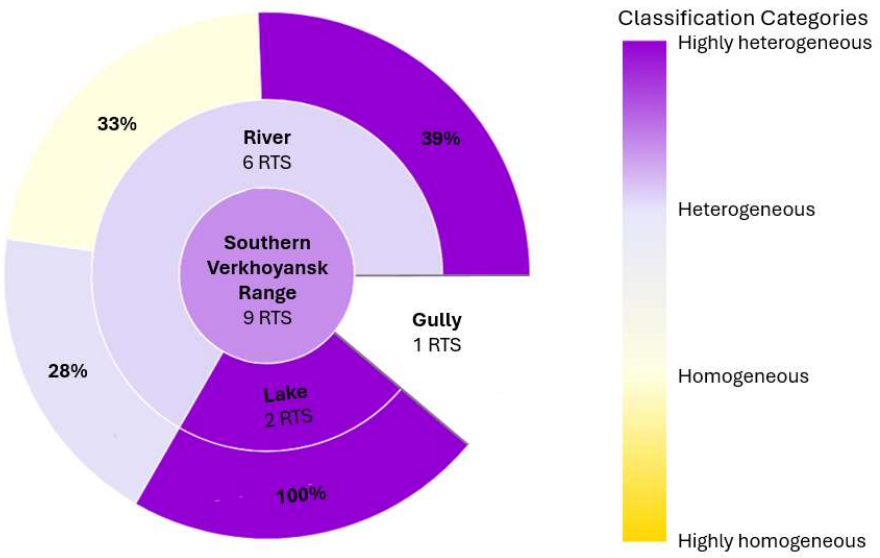


Figure 19 ***Sunburst chart of Southern Verkhoyansk Range, showing the similarity classes corresponding to the terrain positions.*** *Southern Verkhoyansk Range is dominated by the similarity class highly heterogeneous (100% of lake and 39% of river). The three bands of the spectral slope types of each RTS are compared individually between RTSs of the same TP. This, for example, leads to 18 comparisons for the 6 RTSs of TP class river and subsequently percentages of 33% homogeneous, 28% heterogeneous and 39% highly heterogeneous. The TP gully contains only a single RTS and is therefore excluded from the similarity classification. In total, the spectral slopes of the RTS of Southern Verkhoyansk Range are highly heterogeneous within their individual terrain positions.*

Figure 20 shows the sunburst chart of similarity classes corresponding to terrain positions for the study area Chokurdakh. One notable aspect regarding this area is that the implementation of the 5 m inwards buffer resulted in one RTS polygon becoming so small that no spectral slope data could be assigned to it anymore. (This occurred because the ALEX dataset has a spatial resolution of 30 m, while the polygons width was reduced to 8 m.) The lost RTS is not included in any TP class since it can't take part in the spectral slope analysis. The blending approach of the colour code reveals at first glance that the RTSs within Chokurdakh are slightly homogeneous (very light yellow central circle). The TP classes are either coloured in blue (heterogeneous range), in very light yellow (homogeneous range), or in yellow (highly heterogeneous range). A closer look shows that the TP class lake is dominated by the similarity class heterogeneous (73%) while river is dominated by homogeneous (69%), and gully is dominated by highly homogeneous (61%) combined with 33% homogeneous. The lake class contains 69% of the RTS in this area (43 lake RTS versus 6 gully and 13 river RTSs), resulting in a very small general similarity for the entire area. The TP class gully of this area is the first TP class that shows a strong RTS similarity (highly homogeneous).

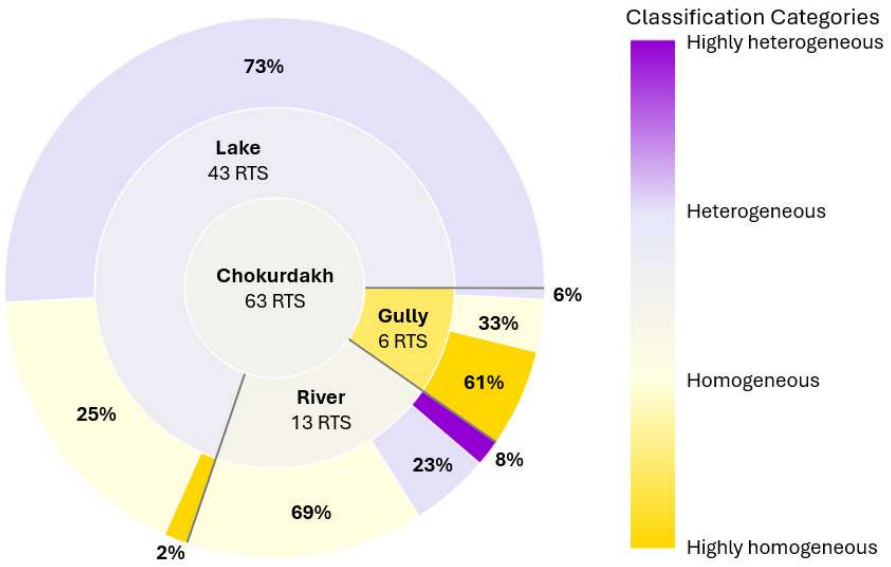


Figure 20: ***Sunburst chart of Chokurdakh, showing the similarity classes corresponding to the terrain positions.*** *The TP class lake dominates the area Chokurdakh. 73% of lake are heterogeneous which results from about half of the similarity comparisons of the whole area. The very homogeneous TP class gully has a strong impact on the average similarity of RTSs at the same TP, which is why the average is slightly homogeneous.*

In Figure 21, the sunburst chart of similarity classes corresponding to terrain positions for study area Iultinsky is displayed. The blending approach of the colour code shows at first glance that Iultinsky shows a little similarity between RTSs within the same TP. The TP classes are either coloured light blue (heterogeneous range) or in yellow (homogeneous and highly homogenerous range). The central circle combines the similarity information for the whole area, it is also coloured light yellow, which shows that the homogeneous classes are dominant. A detailed examination of the TP class lake reveals that it is dominated by the similarity class homogeneous (76%), while 21% of the lake RTS were classified as highly homogeneous and only 2% were classified as heterogeneous. The other yellow TP class is Pond + Gully, of which 70% were classified as homogeneous, 18% as highly homogeneous and 12% as heterogeneous. The other TP classes (Sea, Gully and River) are all dominated by the similarity class heterogeneous.

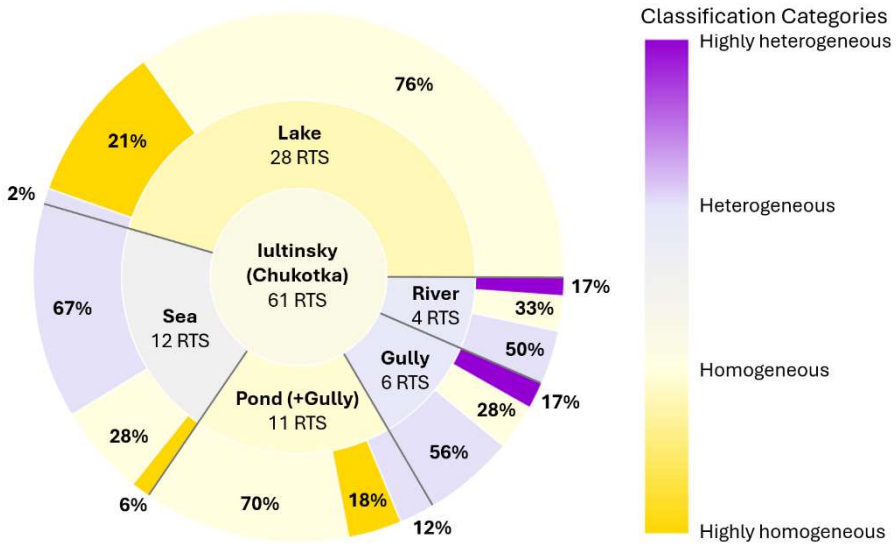


Figure 21: ***Sunburst chart of Iultinsky (Chukotka), showing the similarity classes corresponding to the terrain positions.*** *The area Iultinsky shows in general the similarity class homogeneous for RTS within the TPs lake and pond (+gully). The TP classes sea, river and gully are dominated by the similarity class heterogeneous. Overall homogeneous outweighs heterogeneous.*

To conclude: most RTSs within the same terrain position and the same main area are not very similar towards each other, except for Iultinsky. Because not all areas show an overall homogeneity, sub-hypothesis 3 needs to be rejected.

3.4 Morphology and spectral slope similarity

The extension of the terrain position data frames towards containing morphology information of each RTS (see section 2.4) made it possible to visualize the fraction of RTS per morphology for each study area. The three available morphology classes are thermocirque, thermoterrace, and the combination of both (see Figure 1).

The pie charts of Figure 22 show that the morphology class ‘combination’ dominates the RTSs in Southern Taymyr (70%), Northern Olenek (75%), Chokurdakh (51%) and Iultinsky (57%). Southern Verkhoyansk Range is the only area where the morphology thermocirque is dominating (67%), but it also contains a relevant fraction of the morphology class combination (22%).

Thermocirques also appear in each of the study areas but with very different amounts (6% in Northern Olenek to 67% in Southern Verkhoyansk Rang). The percentage of thermoterrace varies for the study areas Northern Olenek, Chokurdakh, Iultinsky and Southern Verkhoyansk Range from 7% to 33%.

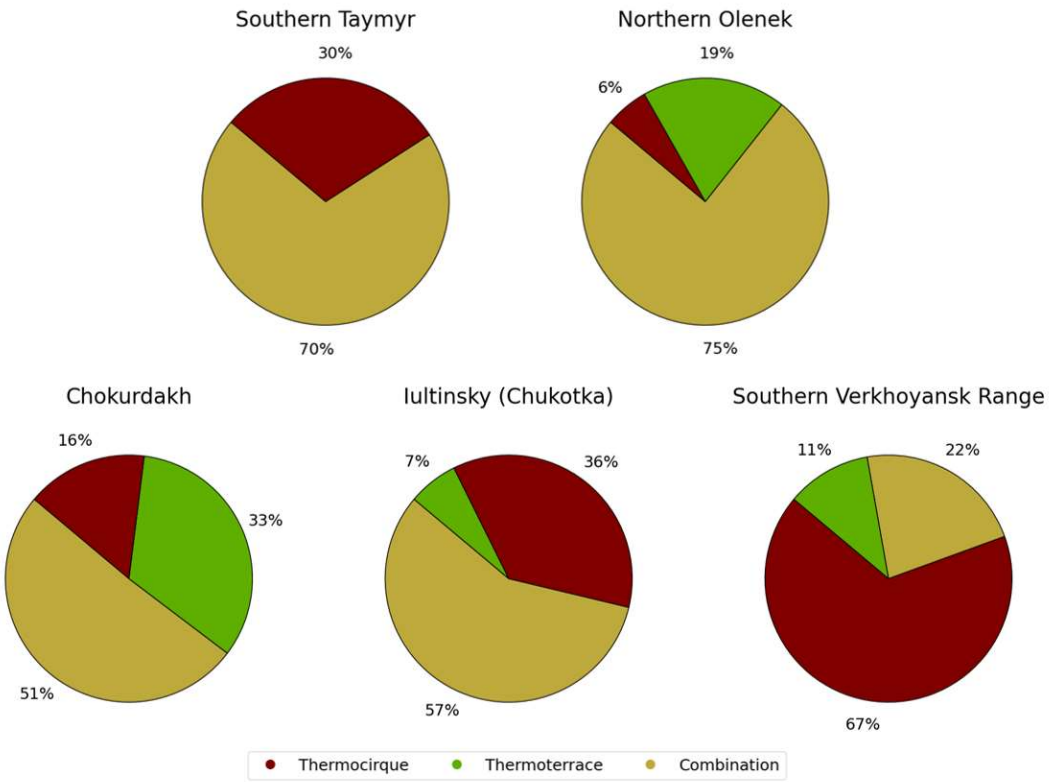


Figure 22 ***Pie charts of percentages of morphology types for each area.*** *The colour of the pie pieces corresponds to the terrain position*

The sub-hypothesis 4 states that the spectral slope of single RTSs that have a certain morphology is always the same as the spectral slope of single RTSs that have the same morphology within the same main area. If this would be completely true, the following plots (Figure 23 to Figure 27) would all be coloured completely yellow because yellow is the colour that represents the class highly homogeneous and the definition of highly homogeneous and sub-hypothesis 4 are basically the same (highly homogeneous: "the sub area is similar to all other sub areas" whereas sub area in this case is replaced with RTSs that have the same morphology.)

The setup of the sunburst charts (Figure 23 to Figure 27) is the same as the one for the sunburst charts of sub-hypothesis 3, the only difference is that the TP classes are replaced with the morphologies.

Figure 23 shows the sunburst chart of similarity classes corresponding to RTS morphology for the study area Iultinsky. The blending approach of the colour code reveals at first glance that RTSs with the same morphology in Iultinsky show similarity. In fact, all three morphology classes and the center circle are coloured yellow (homogeneous range). A closer look shows that all morphology classes are dominated by the similarity class homogeneous (thermocirque 73%, thermoterrace 50%, combination 70%). The similarity

class highly homogeneous is also represented in all morphology classes (thermocirque 13%, thermoterrace 25%, combination 11%). And the similarity class heterogeneous is also present in all morphology classes (thermocirque 17%, thermoterrace 25%, combination 17%).

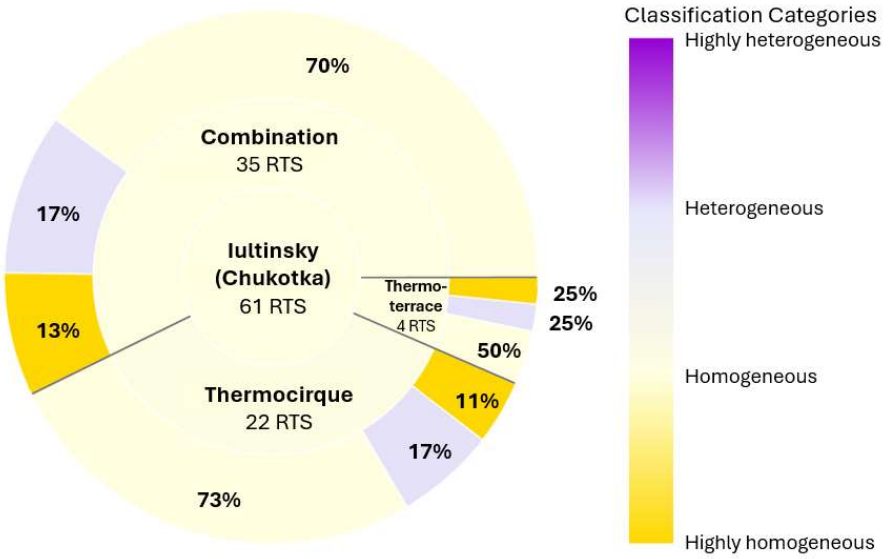


Figure 23: ***Sunburst chart of Iultinsky (Chukotka), showing the similarity classes corresponding to morphology.*** *The dominant similarity class of Iultinsky is homogeneous (thermocirque 73%, thermoterrace 50%, combination 70%). The similarity classes highly homogeneous and heterogeneous are also present in each morphology class. The average similarity for the whole area is homogeneous.*

Figure 24 presents the sunburst chart of similarity classes corresponding to RTS morphology for study area Chokurdakh. The implementation of the 5m inwards buffer resulted in one RTS polygon becoming so small, that no spectral slope data could be assigned to it anymore. The lost RTS is not included in any morphology class since it can't take part in the spectral slope analysis. The blending approach of the colour code shows at first glance that in Chokurdakh RTSs with the same morphology are balanced, neither the class homogeneous nor heterogeneous predominates.. Examining the morphology classes in greater detail shows that thermoterrace and combination are both mostly heterogeneous (thermoterrace 52%, combination 70%). The morphology class thermocirque, however, is mostly homogeneous (57%) and, additionally, even contains the similarity class highly homogeneous (27%).

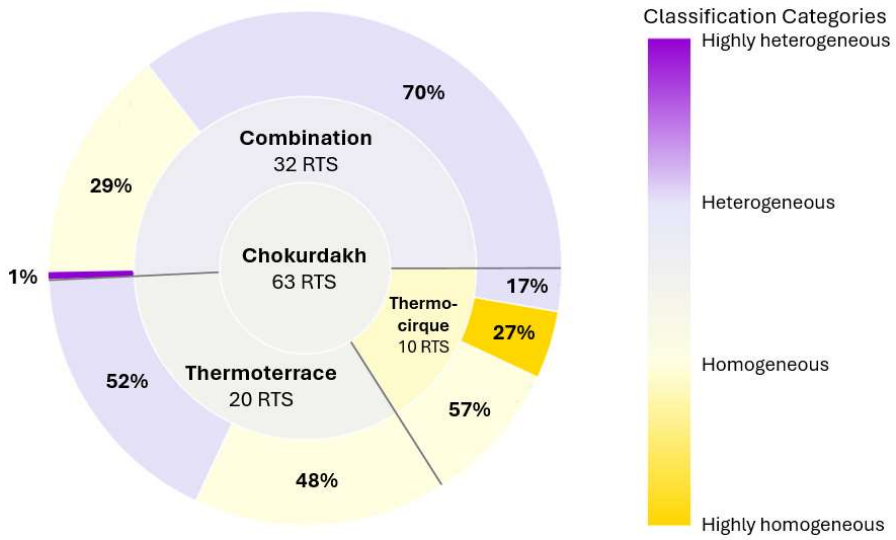


Figure 24: ***Sunburst chart of Chokurdakh, showing the similarity classes corresponding to morphology.*** *In Chokurdakh, the morphology classes thermoterrace and combination have the similarity class heterogeneous (thermoterrace 52%, combination 70%) as largest component. Whereas the largest similarity class within the morphology class thermocirque is homogeneous (57%). Overall, no similarity class predominates this area.*

In Figure 25, the sunburst chart of similarity classes corresponding to morphology for study area Southern Verkhoyansk Range is displayed. The central circle combines the similarity information for the entire area and is coloured light purple, indicating that the heterogeneity classes are dominant. A detailed examination of the morphology classes reveals that the class combination (100% highly heterogeneous) contains only two RTSs which have no similarity to each other in any of the spectral slope types. The morphology class thermocirque contains the similarity classes heterogeneous (67%), homogeneous has 22% and highly heterogeneous 11%.

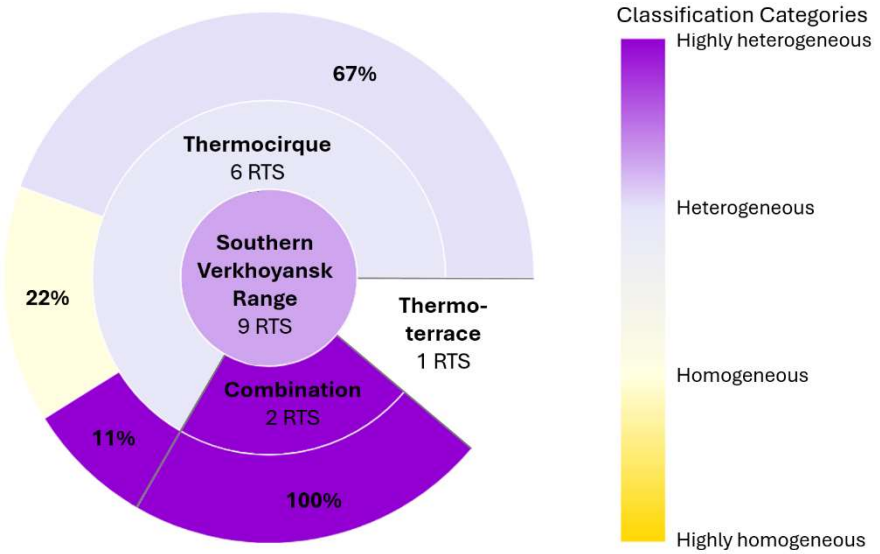


Figure 25: ***Sunburst chart of Southern Verkhoyansk Range, showing the similarity classes corresponding to morphology.*** *Southern Verkhoyansk Range only includes 9 RTSs. The dominating morphology is thermocirque, of which*

67% is classified as heterogeneous. The morphology 'combination' is classified as highly heterogeneous (100%). In total, the RTS of the region are highly heterogeneous within their morphology classes. The morphology thermoterrace contains only a single RTS and is therefore excluded from the similarity classification.

The sunburst chart of similarity classes corresponding to RTS morphology for the study area Southern Taymyr is illustrated in Figure 26. In this study area, there is no predominant similarity class between RTSs with the same morphology. A closer look shows that the largest similarity class within the morphology class thermocirque is homogeneous (62%). On the other hand, the largest similarity class for the morphology class combination is heterogeneous (61%), followed be the similarity class homogeneous (39%).

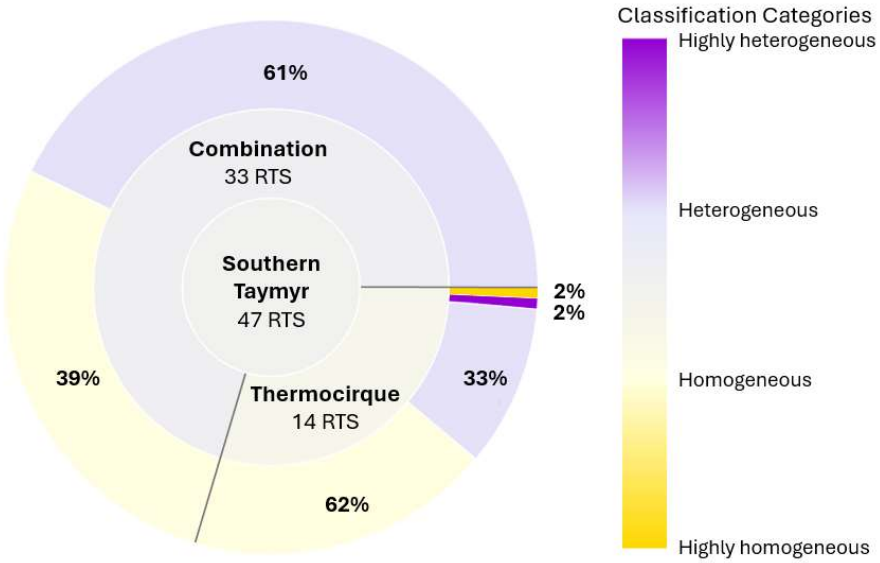


Figure 26: ***Sunburst chart of Southern Taymyr, showing the similarity classes corresponding to morphology.*** *Southern Taymyr includes the two morphology classes thermocirque (62% homogeneous) and 'combination' (61% heterogenous). In total, there are more RTS associated with combination (70%) than with thermocirque, such that on average the RTS within their respective morphology classes are heterogeneous. The slight predominance of heterogeneity is negligible, and the entire area can be considered balanced.*

Figure 27 shows the sunburst chart of similarity classes corresponding to morphology for the study area Northern Olenek. In this study area, RTS are slightly heterogeneous within their morphology. Examining the morphology classes in greater detail shows that thermoterrace and combination are dominated by the similarity class heterogeneous (combination 68%, thermoterrace 83%). The morphology class thermocirque, on the other hand, has the equal similarity classes homogeneous (44%) and highly heterogeneous (44%), with an additional component in the similarity class highly homogeneous (11%).

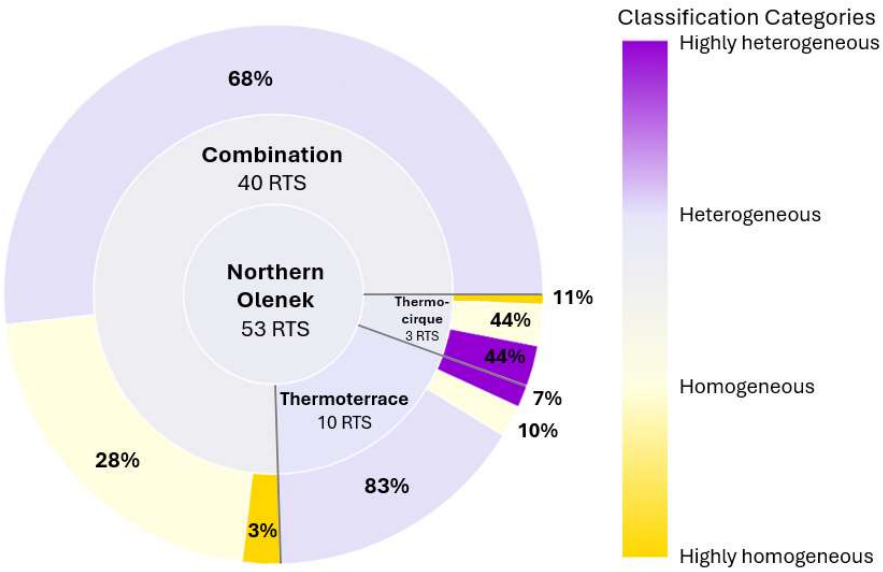


Figure 27: ***Sunburst chart of Northern Olenek, showing the similarity classes corresponding to morphology.*** *The largest fraction of RTSs are in the morphology class combination. 68% of the RTS within this morphology are heterogeneous. The second largest section of the plot represents the morphology thermoterrace which is also mostly heterogeneous (83%). In total, the RTS of Northern Olenek are heterogeneous within their morphology class.*

To conclude: most RTSs with the same morphology and within the same main area are not very similar towards each other. But there are exceptions, notably Iultinsky (Chukotka), where the RTS spectral slope is mostly homogeneous. Since only one area is homogeneous on average, sub-hypothesis 4 needs to be rejected.

3.5 Exploratory insight: terrain position and morphology combinations

A result that emerged during the data analysis, but is not central to this work, are the combinations of the TPs and the morphologies of the RTSs. An interactive sunburst chart representing this dataset can be found at https://zenodo.org/records/15041293 (Heitz 2025). The html file can be downloaded and opened in a local browser. A static representation of the plot is shown in Figure 28 for reference. The reader is encouraged to explore the rich dataset by themselves.

While most RTSs located at the terrain positions lake and sea have a combination morphology, most RTSs located at the TPs gully and pond + gully have a thermocirque morphology. The morphology of RTSs located at rivers varies strongly between the study areas. A common morphology for RTS at rivers could not be identified. The observation that most terrain positions exhibit a specific morphology more frequently than others

supports the theory that terrain position and morphology may be linked. For example, the terrain position could influence erosion and thereby impact the RTS morphology.

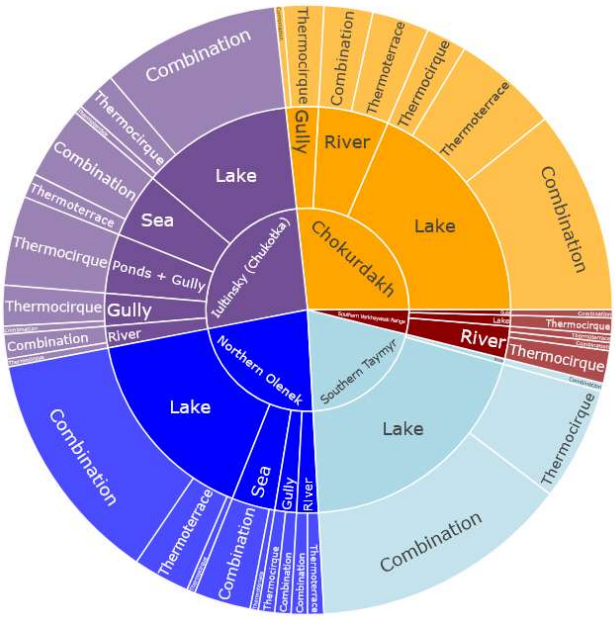


Figure 28: ***Screenshot of the interactive sunburst chart that shows the combinations of terrain positions and morphologies for the RTSs of each study area.*** *Please find an interactive rendition of this dataset at https://zenodo.org/records/15041293 (Heitz 2025). Click on the single study areas or terrain positions to see a close up of a section with adapted fonts for better readability. Hovering over a section reveals additional information, such as the parameter "count" which contains the information on how many RTSs make up that section. The reader is strongly encouraged to experience the richness of the dataset for themselves.*

4 Discussion

The goal of this work is to gain new insights on the variability of RTSs in Siberia. The explored parameters are spectral variability, particularly the spectral slope of tasseled cap indices, the terrain positions and the morphologies. The discussion is organized around several external parameters, with each section following a structure similar to that of the methods (section 2) and results (section 3), grouped into comparisons between RTS at different levels of spatial detail.

4.1 Impact of a 5 m inward buffer towards the RTS polygons

The analysis of the comparison of the spectral slope of the sub areas was conducted two times, one time including all values of each RTS geometry. Another time with a 5 m inward buffer applied to each RTS geometry (see section 2.2). This inward buffer counteracted a 5 m buffer applied to each RTS during the initial mapping. One would expect that merging data from the buffer areas with the data from the RTS themselves, incorporating the initial 5 m buffer, might change the outcome. The comparison of the results (Figure 14 and Figure 15, section 3.2) show the impact of the 5 m buffer on the spectral slope similarity in three of the five areas. When the buffer area is removed in Southern Taymyr, the spectral slope similarity becomes more homogeneous. The area Iultinsky shows a strong change in the similarity class distribution (decrease of highly heterogeneous and of homogeneous and increase of heterogeneous). In contrast, the results in Northern Olenek remain mostly stable, with only a minor change in one similarity class. Further analysis shows, that the area within the 5 m buffer zone can reach over 60% of the total area of one RTS, depending on its individual size and geometry. A detailed analysis of how much of the initial RTS geometry area is actually part of the buffer is presented in Appendix C3.

Since the results of the statistical analysis is influenced by the inclusion of spectral slope values from the buffer regions, that aren't part of the RTS themselves, all following analyses have been performed with buffer-corrected RTS geometries (sub-hypothesis 3 and 4).

4.2 Spectral slope trends

Figure 12 of section 3.1 is a violin plot of the cumulative spectral slope (net increase or decrease over time) of each area, for each of the tasseled cap indices. The strongest changes are observed for the greenness index of Southern Taymyr, Northern Olenek and Iultinsky. All three areas experience increased greenness, which could suggest a stabilisation of the majority of the RTS areas since a stabilisation comes along with vegetation growth. The stabilisation mechanism is described in C.R. Burn and P.A. Friele 1989. In short, if the meltwater disappears, the mudflow stops. Non-moving and nutrient rich mud promotes vegetation growth. The mean values for the other spectral slope types (brightness and wetness), on the other hand, experience no significant changes in the observed time frame. This is somewhat surprising, if the increase in greenness is indeed a sign for stabilisation. During the stabilisation process, the wetness and brightness indices should decrease due to less melt water availability, less mud movement and the connected vegetation growth.

Southern Verkhoyansk Range, on the other hand, shows an increase of brightness and a decrease of greenness. These changes could indicate a new disruption of the vegetation that could be the result of the reinitialization (polycyclic nature) of the RTSs. The very small increase in wetness, however, does not support this theory. A reinitialization would be the result of further melting of massive ground ice which would increase the wetness index of the RTSs when new melt water/mud streams form. Moreover, the means of the spectral slope distributions of Southern Verkhoyansk Range show only minor changes, compared to the means of the other areas. The spectral slope distributions of the Chokurdakh area are very similar to those of Southern Verkhoyansk Range.

The comparison of spectral slopes of single RTSs located within one area, either with respect to shared terrain position (sub-hypothesis 3) or shared morphology (sub-hypothesis 4) is informative as well. On the surface, the result of both analyses is, that neither the terrain position nor the morphology directly impact or relate to the RTS spectral slope similarities. Therefore, sub-hypothesis 3 and 4 (cf. Table 2) are rejected.

However, it appears that the spectral slope similarity trend across all areas is similar for both the TP and the morphology analyses. If an area shows a lot of similarity (e.g. Iultinsky), a medium amount of similarity (e.g. Chokurdakh) or nearly no similarity (e.g.

Southern Verkhoyansk Range), it will do so in both analyses. This parallel behaviour can have several reasons, one could be that the terrain position and the morphology might be strongly linked to each other. A strong link in between terrain position and morphology could, for example, exist for the terrain types gully and pond+gully. Both of these terrain positions appear strongly related to the morphology thermocirque. Possibly, there could be process based relation, e.g. erosion, between the terrain positions and the morphologies.

Another explanation for such a link may be human error in the RTS mapping process. It might be, that areas with higher similarity scores contained more precisely drawn RTS polygons. Inspection of Validation Dataset 2 revealed that the 5m buffer, that was applied to each polygon, was added with varying generosity, examples can be observed in Appendix C6, Figure 34. This mismatch impacts the spectral slope data and, therefore, the similarity of the spectral slope data. The area that shows the most similarity between the RTSs for both the TP and the morphology is Iultinsky. This is also the area where the 5 m inward buffer shows the strongest effect per RTS polygon. 49% of the RTSs in the Iultinsky area lose more than 20% of their area due to the buffer (see Appendix C3). Such a large proportion of buffer area has a strong effect on the similarity analysis through wrongly included values.

4.3 Range of variation in spectral slope, terrain position and morphology

In general, the parameters spectral slope, terrain position and morphology span a large range. The spectral slope of one RTS compared to another can take all values from completely similar to not similar at all (divided into 4 different similarity classes by definition, see Figure 8). However, not all of the study areas have similarity values across the full range within one terrain position or morphology class. Some classes show only one or two similarity classes. Examples are the terrain position class lake of Southern Verkhoyansk Range, which is 100% highly heterogeneous, or the morphology class thermoterrace of Chokurdakh, which only includes the similarity classes homogeneous (48%) and heterogeneous (52%).

The terrain positions exhibit a high variability as well. All five TP classes appear together only once in the study area Iultinsky. In contrast, the smallest amount of TP classes is

found in Southern Taymyr with lake and river. A special finding is that the TP class *pond + gully* is found only in the study area Iultinsky (see section 3.3).

In contrast, all study areas except for Southern Taymyr contain RTSs of all three morphology classes. Southern Taymyr has RTSs in only two morphology classes.

4.4 RTS spectral slope versus geographical setting

The structure of validation dataset 2 (section 1.2.1), with study areas spread across the entirety of Siberia (e.g. see Figure 11) suggests that there might be a geographical dependence of the spectral slope. Indeed, one might expect that the geographical setting, including local climate, influences the thawing of permafrost and RTS formation. The investigation of combined spectral slopes accumulated over each study area, such as presented in Figure 11, relate to sub-hypothesis 1: if indeed the geographical position influences the spectral slope the combined spectral slope of the study areas located in vastly different environments should share no similarity with one another.

Further evidence for a geographical influence on the spectral slope may be found in the parallel results for morphology and terrain positions dependent spectral slope analysis (section 4.2). Since both morphology and terrain position are part of the geographical setting, similar results for each of the areas may indicate a link.

The influence of the geographical setting might appear in the relationship between the mean spectral slope of the study areas compared to the geospatial position (latitude resp. longitude), since the geospatial position is an important aspect of the geographical setting. The plots of the mean spectral slope versus geospatial position are found as Figure 31 and Figure 32 in Appendix C4 However, the spectral slope means of the areas are uncorrelated with both latitude and longitude. On the one hand, this lack of correlation could indicate no relation between the combined spectral slope and the geographical setting. On the other hand, the assumed relationship between geospatial position and geographical setting may be highly oversimplified. Indeed, since the geographical setting includes many more parameters that should be strongly linked to RTS initiation and development, the assumption cannot be considered disproven. Some examples of relevant parameters are: ice content, ice development/history, deposit type/soils and local geology.

5 Conclusion and Outlook

Within this study, it has been shown that the RTS in multiple study areas across Siberia are very variable. The spectral slope of three tasseled cap indices, recorded over a period of 20 years, was the observed variable. The RTS spectral slopes have been compared with one another within the study areas, as well as, correlated with other RTS properties terrain position and RTS morphology. Additionally, the combined spectral slopes of the study areas have been compared.

Indeed, the RTS within this study have been found to be highly variable across all parameters. Therefore, the main hypothesis is accepted (cf. Table 2).

To quantify the variability, a similarity classifier with a four-step scale (highly heterogeneous, heterogeneous, homogeneous and highly homogeneous) based on statistical comparison of spectral slope values has been introduced. This classification scheme enabled the statistical similarity analysis of RTS on differing spatial scale. No significant similarities were found comparing the combined spectral slope of the different main areas. A simple geospatial correlation was not found either. Therefore, sub-hypothesis 1 was accepted. The lack of significant similarity in cross-study-area results indicates that the spectral slope variability on this scale overshadows similarity trends or patterns.

Comparisons of the combined spectral slope of sub-areas within the main areas supported the hypothesis of high variability of sub-area combined spectral slope within the main areas. However, since a few significant correlations could be found, sub-hypothesis 2 is partly rejected.

The analysis of sub-hypothesis 3 and 4 showed that the parameters terrain position and morphology differ across the RTSs of the study areas. Moreover, both analyses widened the understanding of the variability of spectral slope regarding the level of singe RTS comparisons within the study areas. A summary of the hypothesis is presented in Table 2.

Table 2: Overview of hypothesis acceptance/rejection

Main hypothesis	Retrogressive thaw slumps within the study areas show differences in one or more parameters.	Accepted
Sub-hypothesis 1	The combined spectral slope of the RTSs in one area is never the same as the combined spectral slope of the RTSs in any of the other areas.	Accepted
Sub-hypothesis 2	The spectral slope of all the RTSs in one selected sub-area is never the same as the spectral slope of all the RTSs in any of the other sub-areas within the same main area.	Partly rejected
Sub-hypothesis 3	The spectral slope of single RTSs that are located in a certain terrain position is always the same as the spectral slope of single RTSs that are located at the same terrain position within the same main area.	Rejected
Sub-hypothesis 4	The spectral slope of single RTSs that have a certain morphology is always the same as the spectral slope of single RTSs that have the same morphology within the same main area.	Rejected

In conclusion, the observed spectral variability of retrogressive thaw slumps across Siberia can be summarised: larger areas that are located distant from each other don't show similarities regarding the spectral slope. Smaller areas located closer to each other are mostly different from each other, although a few similarities between some of the areas could be found. Neither the terrain position nor the morphology directly relate to the similarities that were found regarding the spectral slope. The result of the analysis is that the spectral slope of RTSs is a property with high variability that requires further study.

One promising path to further understand the variability of retrogressive thaw slumps would be to characterize RTSs according to their process stages (initiation, active phase, stabilization) and to compare those to the spectral slope data. Since the spectral slope data shows how the landscape changes, this process-based classification could be a

relatively strong tool to distinguish different RTS stages from other landscape changes. Possibly, a fingerprint system that combines the three slope type values could be developed. Furthermore, other parameters like RTS volume or dynamics could also be correlated with spectral variability.

A weakness discovered in this work, that needs to be addressed in future studies, is the human bias included in the manual assignment of the morphology classes. For example, morphometrical parameters like an RTS length to width ratio could be promising. A “roundness” factor, that would need to be developed, might be used as a human bias free tool to determine which morphology class a RTS belongs to.

6 References

C.R. Burn and P.A. Friele (1989): Geomorphology, Vegetation Succession, Soil Characteristics and Permafrost in Retrogressive Thaw Slumps near Mayo, Yukon Territory. In: *Arctic* (42), S. 31–40.

Colaboratory-team (2023): Google Colab. Version 1.1.0: Google. Online verfügbar unter https://colab.research.google.com/github/.

Colaboratory-team (2024): Google Colab. Version 1.2.0: Google. Online verfügbar unter https://colab.research.google.com/github/.

Esri, Maxar, Earthstar Geographics, and the GIS User Community: World Imagery. Online verfügbar unter https://services.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer, zuletzt geprüft am 26.02.2025.

Fraser, Robert; Olthof, Ian; Kokelj, Steven; Lantz, Trevor; Lacelle, Denis; Brooker, Alexander et al. (2014): Detecting Landscape Changes in High Latitude Environments Using Landsat Trend Analysis: 1. Visualization. In: *Remote Sensing* 6 (11), S. 11533–11557. DOI: 10.3390/rs61111533.

Heitz, Julia (2025): Combinations of terrain positions and morphologies for retrogressive thaw slumps of different study areas across Siberia.

Jones, Benjamin M.; Grosse, Guido; Farquharson, Louise M.; Roy-Léveillée, Pascale; Veremeeva, Alexandra; Kanevskiy, Mikhail Z. et al. (2022): Lake and drained lake basin systems in lowland permafrost regions. In: *Nat Rev Earth Environ* 3 (1), S. 85–98. DOI: 10.1038/s43017-021-00238-9.

Kirchner, James (2001): Data Analysis Toolkit #5: Uncertainty Analysis and Error Propagation. University Berkeley. Online verfügbar unter https://seismo.berkeley.edu/~kirchner/eps_120/Toolkits/Toolkit_05.pdf, zuletzt geprüft am 25.03.2025.

Kruchten, N., Seier, A., Parmer, C. (2024): Plotly PY. An interactive, open-source, and browser-based graphing library for Python (Version 5.24.1): Zenodo.

Leibman, Marina; Nesterova, Nina; Altukhov, Maxim (2023): Distribution and Morphometry of Thermocirques in the North of West Siberia, Russia. In: *Geosciences* 13 (6), S. 167. DOI: 10.3390/geosciences13060167.

Mackay, J. Ross (1966): Segregated Epigenetic Ice and Slumps in Permafrost. Mackenzie Delta Area, NWT: Geographical Bulletin (8).

McDonald, John (2014): Handbook of Biological Statistics. 3. Aufl. Baltimore, Maryland, U.S.A.: SPARKY HOUSE PUBLISHING.

Mohd Razali, Nornadiah and Yap, Bee (2011): Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests. In: *J. Stat. Model. Analytics* (2), Artikel 1, S. 21–33.

Nesterova, Nina; Leibman, Marina; Kizyakov, Alexander; Lantuit, Hugues; Tarasevich, Ilya; Nitze, Ingmar et al. (2024): Review article: Retrogressive thaw slump characteristics and terminology. In: *The Cryosphere* 18 (10), S. 4787–4810. DOI: 10.5194/tc-18-4787-2024.

Nitze, Ingmar; Grosse, Guido (2016): Detection of landscape dynamics in the Arctic Lena Delta with temporally dense Landsat time-series stacks. In: *Remote Sensing of Environment* 181, S. 27–41. DOI: 10.1016/j.rse.2016.03.038.

Nitze, Ingmar; Grosse, Guido; Jones, Benjamin; Arp, Christopher; Ulrich, Mathias; Fedorov, Alexander; Veremeeva, Alexandra (2017): Landsat-Based Trend Analysis of Lake Dynamics across Northern Permafrost Regions. In: *Remote Sensing* 9 (7), S. 640. DOI: 10.3390/rs9070640.

Nitze, Ingmar; Grosse, Guido; Jones, Benjamin M.; Romanovsky, Vladimir E.; Boike, Julia (2018): Remote sensing quantifies widespread abundance of permafrost region disturbances across the Arctic and Subarctic, Datasets.

Nitze, Ingmar; Heidler, Konrad; Nesterova, Nina; Küpper, Jonas; Schütt, Emma; Hölzer, Tobias et al. (2024a): DARTS: Multi-year database of AI detected retrogressive thaw slumps (RTS) and active layer detachment slides (ALD) in hotspots of the circum-arctic permafrost region - v1.

Nitze, Ingmar; Lübker, Tillmann; Grosse, Guido (2024b): Pan-Arctic Visualization of Landscape Change (2003-2022), Arctic PASSION Permafrost Service.

Obu, Jaroslav; Westermann, Sebastian; Kääb, Andreas; Bartsch, Annett (2018): Ground Temperature Map, 2000-2016, Northern Hemisphere Permafrost. Unter Mitarbeit von Alfred Wegener Institute, Helmholtz Centre for Polar and Marine Research, Bremerhaven.

Pekel, Jean-François; Cottam, Andrew; Gorelick, Noel; Belward, Alan S. (2016): High-resolution mapping of global surface water and its long-term changes. In: *Nature* 540 (7633), S. 418–422. DOI: 10.1038/nature20584.

QGIS Development Team (2023): QGIS Geographic Information System. Prizren. Version 3.34: Open Source Geospatial Foundation. Online verfügbar unter http://qgis.osgeo.org.

Sayre, Roger; Noble, Suzanne; Hamann, Sharon; Smith, Rebecca; Wright, Dawn; Breyer, Sean et al. (2019): A new 30 meter resolution global shoreline vector and associated global islands database for the development of standardized ecological coastal units. In: *Journal of Operational Oceanography* 12 (sup2), S47-S56. DOI: 10.1080/1755876X.2018.1529714.

Shur, Yuri; Jorgenson, M. Torre; Kanevskiy, M. Z. (2011): Permafrost. In: Vijay P. Singh, Pratap Singh und Umesh K. Haritashya (Hg.): Encyclopedia of Snow, Ice and Glaciers. Dordrecht: Springer Netherlands (Encyclopedia of Earth Sciences Series), S. 841–848.

Wu, Qiusheng (2020): geemap: A Python package for interactive mapping with Google Earth Engine. In: *JOSS* 5 (51), S. 2305. DOI: 10.21105/joss.02305.

Appendix A: German Summary (Deutsche Zusammenfassung)

Durch den Einfluss des Klimawandels tauen die Permafrost-Regionen der Erde. Das Tauen von Permafrostböden geht häufig mit starken Landschaftsveränderungen und dem Austritt von Treibhausgasen einher. Eine Art dieser durch das Tauen von Permafrost bedingter Landschaftsveränderungen sind *retrogressive thaw slumps (RTS),* auf Deutsch *thermokarstische Rückzugsnischen*. Das Interesse an RTSs steigt wegen ihrer großen Bedeutung für das lokale Landschaftsbild und ihrem Einfluss auf globale Klimaänderungen (Nesterova et al. 2024). Um das wissenschaftliche Verständnis bezüglich RTSs zu erweitern, beschäftigt sich diese Arbeit mit verschiedenen Eigenschaften von RTSs. Der Fokus liegt dabei auf der spektralen Variabilität von RTS, konkret dem *spectral slope,* welcher die gemittelte Änderung eines spektralen Indexes über eine bestimmte Zeitspanne darstellt. Die Datensätze, mit denen der Hauptteil der Analyse durchgeführt wurde, sind: Pan-Arctic Visualization of Landscape Change (2003-2022) und das Validation Dataset 2. Die Fünf Untersuchungsgebiete mit einer Größe von jeweils 10.000 km^2 liegen über Sibirien verteilt, jedes dieser Gebiete enthält 10 zufällig positionierte Untergebiete von je 100 km^2, innerhalb der Untergebiete sind die Umrisse der RTSs bekannt. Der spectral slope der RTSs wurde zwischen den verschiedenen Untersuchungsgebieten und, darüber hinaus, zwischen den Untergebieten innerhalb der Untersuchungsgebiete verglichen. Zusätzlich wurden die RTSs einzeln nach ihrer Lage im Gelände und nach ihrer Morphologie klassifiziert. RTSs innerhalb derselben Untersuchungsgebiete und derselben Geländeposition bzw. Morphologieklasse wurden ebenfalls hinsichtlich ihrer spectral slopes verglichen. Die Ergebnisse zeigen eine deutliche Variation der spectral slopes der verschiedenen Untersuchungsgebiete, ohne signifikante Ähnlichkeiten. Während auf einer kleineren räumlichen Ebene die verschiedenen Untergebiete innerhalb desselben Untersuchungsgebiets überwiegend heterogen sind, wurden einige signifikante Ähnlichkeiten in Bezug auf ihre spectral slopes festgestellt. Der Vergleich einzelner RTSs, welche sich innerhalb desselben Untersuchungsgebiets in derselben Geländeposition befanden oder dieselbe Morphologie aufwiesen, zeigte keine Korrelation zwischen spectral slope Ähnlichkeit und der Geländeposition oder Morphologie. Zusätzlich konnten Erkenntnisse über die Häufigkeit der verschiedenen Morphologien in Zusammenhang mit verschiedenen Geländepositionen gewonnen werden. In 4 von 5 Gebieten befinden sich die

überwiegende Anzahl der RTSs in der Geländeposition „Seeufer“, und die meisten von ihnen weisen eine kombinierte Morphologie auf, die Merkmale sowohl von thermocirque als auch von thermoterrace RTS enthält. Das wichtigste Ergebnis dieser Studie ist, dass der spectral slope eine Eigenschaft ist, die in Bezug auf RTSs stark variabel ist. Es wurde ein Schema zur Klassifizierung der RTS-Ähnlichkeit entwickelt. Dieses Klassifizierungsschema kann auch für den Vergleich anderer RTS-Eigenschaften modifiziert werden.

Appendix B: Declaration of authorship

I hereby certify that the thesis I am submitting is entirely my own original work, except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works by any other authors, in any form, is properly acknowledged.

26.03.2025, Julia Heitz

Date, Signature

Appendix C: Further tables and figures

Appendix C1: Table citing each Esri World Imagery Basemap image used.

Table 3: Esri image citations. Directly accessed via Esri World Imagery wayback. The coordinates refer to the WGS 84 projection. Resolution and accuracy are the same for each image. Resolution: *pixels in the source image represent a ground distance of 1.2 meters.* Accuracy: *objects displayed in this image are within 5 meters of true location.*

Coordinates (WGS 84)	Citation
x: 97.1656 y: 72.8032	Maxar (WV02) image captured on **Jul 28, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 97.7626 y: 72.8824	Maxar (WV03) image captured on **Jul 7, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 97.5154 y: 72.6192	Maxar (WV02) image captured on **Jul 28, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 97.9432 y: 72.6325	Maxar (WV03) image captured on **Jul 7, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 98.4411 y: 72.7143	Maxar (WV03) image captured on **Jul 7, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 97.1303 y: 72.4493	Maxar (WV03) image captured on **Sep 7, 2016** as shown in the **2025-01-30** version of the World Imagery map.
x: 97.8774 y: 72.4156	Maxar (WV02) image captured on **Jul 20, 2019** as shown in the **2025-01-30** version of the World Imagery map.
x: 116.5239 y: 73.6417	Maxar (WV02) image captured on **Jul 30, 2023** as shown in the **2025-01-30** version of the World Imagery map.
x: 116.6242 y: 73.5335	Maxar (WV03) image captured on **Jul 4, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 118.0844 y: 73.5545	Maxar (WV03) image captured on **Jun 8, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 117.5639 y: 73.2330	Maxar (WV02) image captured on **Jun 29, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 116.1827 y: 73.0741	Maxar (WV03) image captured on **Jul 4, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 145.1977 y: 71.5928	Maxar (WV02) image captured on **Jun 30, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: 146.5190 y: 71.5033	Maxar (WV03) image captured on **Aug 10, 2022** as shown in the **2025-01-30** version of the World Imagery map.

x: 147.1885 y: 71.5371	Maxar (WV02) image captured on **Jun 12, 2019** as shown in the **2025-01-30** version of the World Imagery map.
x: 145.6328 y: 71.2914	Maxar (WV02) image captured on **Jun 15, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: 147.1210 y: 71.1897	Maxar (WV03) image captured on **Jun 17, 2019** as shown in the **2025-01-30** version of the World Imagery map.
x: 144.6616 y: 71.0933	Maxar (WV02) image captured on **Aug 9, 2021** as shown in the **2025-01-30** version of the World Imagery map.
x: -174.1289 y: 67.0583	Maxar (WV02) image captured on **Aug 6, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: -173.8252 y: 66.9928	Maxar (WV03) image captured on **Aug 20, 2021** as shown in the **2025-01-30** version of the World Imagery map.
x: -173.9055 y: 66.7161	Maxar (WV02) image captured on **May 30, 2022** as shown in the **2025-01-30** version of the World Imagery map.
x: -174.0042 y: 66.4336	Maxar (WV02) image captured on **Aug 30, 2023** as shown in the **2025-01-30** version of the World Imagery map.
x: -174.7740 y: 66.8132	Maxar (WV02) image captured on **Aug 13, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: -175.1340 y: 66.9668	Maxar (GE01) image captured on **Jul 14, 2021** as shown in the **2025-01-30** version of the World Imagery map.
x: -175.4439 y: 66.6612	Maxar (GE01) image captured on **Jul 14, 2021** as shown in the **2025-01-30** version of the World Imagery map.
x: -175.8431 y: 66.8907	Maxar (WV03) image captured on **Jun 16, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: 131.5629 y: 63.9987	Maxar (WV03) image captured on **Jun 30, 2020** as shown in the **2025-01-30** version of the World Imagery map.
x: 131.7427 y: 63.8514	Maxar (GE01) image captured on **Jun 2, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 130.3086 y: 63.7926	Maxar (WV02) image captured on **Jun 5, 2024** as shown in the **2025-01-30** version of the World Imagery map.
x: 131.7366 y: 63.3009	Maxar (GE01) image captured on **Jun 2, 2024** as shown in the **2025-01-30** version of the World Imagery map.

Appendix C2: Histograms of spectral slope with area as y-axis plotted side-by-side.

The single histograms were plotted on the same y-axis (Figure 29) in units of area. This enables the comparison of both the spectral slope distributions and the affected area. Chokurdakh and Northern Olenek have the largest RTS covered area. Iultinsky (Chukotka) has an intermediate amount of RTS area. The least RTS covered areas are found in the Southern Taymyr and Southern Verkhoyansk Range areas. It is important to remember, that the affected area corresponds to the integral over the distribution, such that the distribution width and peak height are equally important. Generally, the distributions either have their maximum at 0 (which means no change) or in the positive value range (which indicates an increase in tasseled cap index). Only the maximum of greenness of Southern Verkhoyansk Range and Chokurdakh, as well as, the maximum brightness of Northern Olenek are in the negative value range.

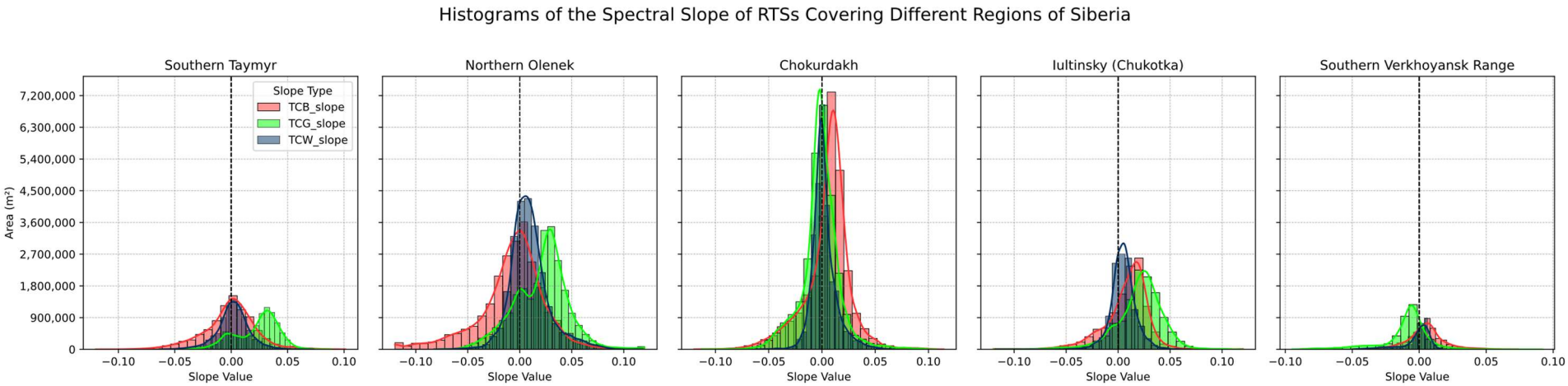


Figure 29: ***Histograms of spectral slope with area as y-axis.*** *Shown next to each other for better comparability. Note that the y-axis is in units of area.*

Appendix C3: Percentage of RTS areas affected by the 5 m buffer

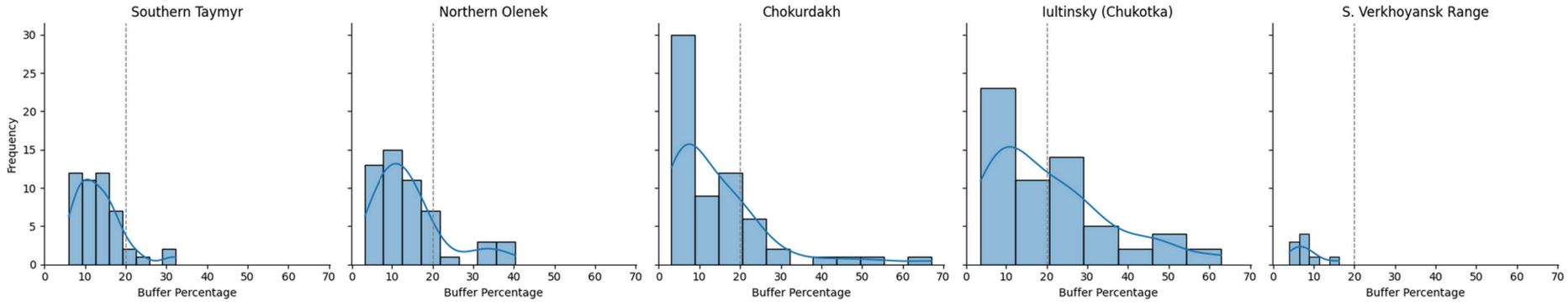


Figure 30: ***Histograms showing the percentage of the RTS area affected by the 5 m inward buffer.*** *The dashed vertical line at 20% helps identify RTSs where more than 20% of the area of the individual RTS is affected by the buffer. RTSs in the area Iultinsky are most affected by the 5 m buffer.*

The 5 m buffer zone applied to each RTS geometry can be a substantial area fraction of each RTS geometry. Figure 30 shows histograms of the buffer area fraction for each RTS within the different study areas. It is notable, for example, that 49% of the RTSs in Iultinsky shrink more than 20% in size. The fraction of RTSs affected by more than 20% in the other study areas are: Southern Taymyr = 11%, Northern Olenek = 15%, Chokurdakh = 17%, Southern Verkhoyansk Range = 0%.

Appendix C4: Correlation of the mean spectral slope of the study areas as a function of latitude and longitude.

The coloured background area shown within the Figure 32 and Figure 31 visualize the uncertainty in the estimated regression lines themselves. The confidence interval bounds, set at 95%, reflect the range within which the regression line is likely to fall. All confidence intervals are large enough that each corresponding regression line could be shown as horizontal line within the boundaries, which means that the trend shown by the regression line has the same probability to be true as no trend at all (horizontal line). This shows that no significant correlation was found.

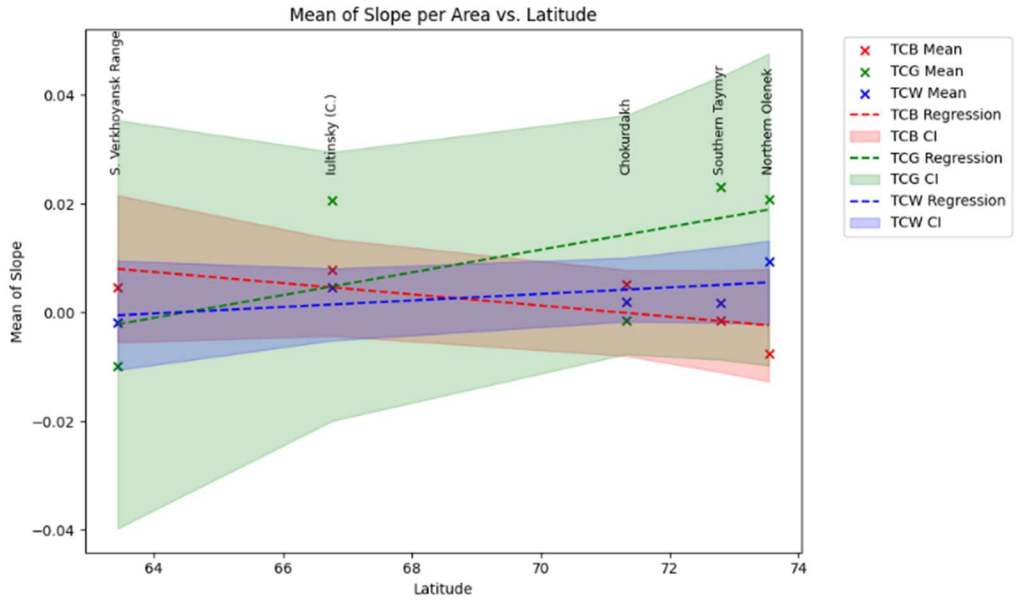


Figure 31: ***Correlation of the means of the spectral slope types per area and latitude.*** *No trend can be found.*

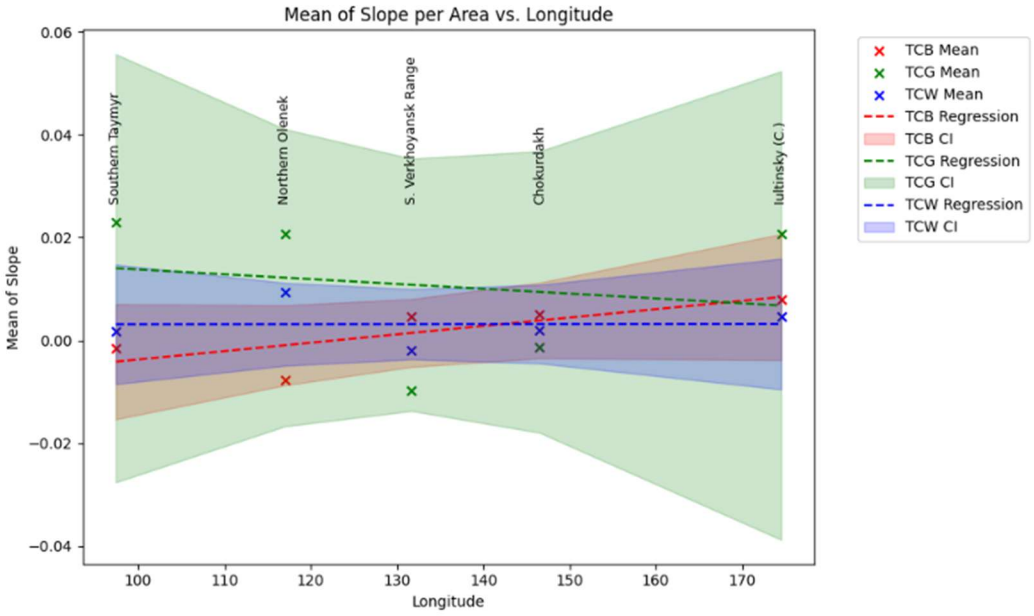


Figure 32: ***Correlation of the means of the spectral slope types per area and longitude.*** *No trend can be found.*

Appendix C5: Data preparation for sunburst visualization and associated rounding error estimates

$$\boldsymbol{Count} = \frac{S.Percentage\ [\%] \times Total\ count\ of\ class\ values}{100[\%]}$$

Error propagation in arithmetic calculations, multiplication and division formula:

$$\Delta z = \sqrt{\left(\frac{\Delta x}{x}\right)^2 + \left(\frac{\Delta y}{y}\right)^2} \cdot z$$

x, y = Measured quantities
$\Delta x, \Delta y$ = Uncertainties of quantities
z = Quantity that combines x and y

Applied formula:

$$\boldsymbol{Count\ error} = \frac{S.Percentage\ error[\%]}{S.Percentage[\%]} \cdot count$$

Note: The *S.Percentage error* is ±0.5% in all cases, since it results from rounding S.Percentage to 0 decimal places.

Figure 33: ***Formulars that were used to calculate Count and Count error****. Image compiled as summary of the relevant parts in (Kirchner 2001).*

Table 4: Rounding errors for the pie piece size calculation in terrain position related sunburst charts (Figure 17 to Figure 21). S.Percentage is the abbreviation of similarity percentage, which was rounded to 0 decimal places. This rounding results in a +/- 0.5% error. Propagating this error results in a chart section specific count error. The size of the individual sections of the sunburst chart was derived from Count. The count error describes the error in the visual size of each section. Importantly, the count errors apply to the visualisation only and do not influence the statistical analysis in this work.

Region	Terrain position	Classification	S.Percentage [%]	S.Percentage error [%]	Count	Count error
Northern Olenek	Sea	Homogeneous	17	±0.5	1.36	±0.04
Northern Olenek	Sea	Heterogeneous	71	±0.5	5.68	±0.04
Northern Olenek	Sea	Highly heterogeneous	12	±0.5	0.96	±0.04
Iultinsky (Chukotka)	Sea	Highly homogeneous	6	±0.5	0.72	±0.06
Iultinsky (Chukotka)	Sea	Homogeneous	28	±0.5	3.36	±0.06
Iultinsky (Chukotka)	Sea	Heterogeneous	67	±0.5	8.04	±0.06
Southern Taymyr	Lake	Highly homogeneous	1	±0.5	0.46	±0.23
Southern Taymyr	Lake	Homogeneous	56	±0.5	25.76	±0.23
Southern Taymyr	Lake	Heterogeneous	43	±0.5	19.78	±0.23
Northern Olenek	Lake	Highly homogeneous	5	±0.5	1.85	±0.19
Northern Olenek	Lake	Homogeneous	30	±0.5	11.1	±0.18
Northern Olenek	Lake	Heterogeneous	66	±0.5	24.42	±0.19
Chokurdakh	Lake	Highly homogeneous	2	±0.5	0.88	±0.22
Chokurdakh	Lake	Homogeneous	25	±0.5	11.0	±0.22
Chokurdakh	Lake	Heterogeneous	73	±0.5	32.12	±0.22
Iultinsky (Chukotka)	Lake	Highly homogeneous	21	±0.5	5.88	±0.14

Iultinsky (Chukotka)	Lake	Homogeneous	76	±0.5	21.28	±0.14
Iultinsky (Chukotka)	Lake	Heterogeneous	2	±0.5	0.56	±0.14
Southern Verkhoyansk Range	Lake	Highly heterogeneous	100	±0.5	2.0	±0.01
Northern Olenek	River	Heterogeneous	100	±0.5	4.0	±0.02
Chokurdakh	River	Homogeneous	69	±0.5	8.97	±0.06
Chokurdakh	River	Heterogeneous	23	±0.5	2.99	±0.06
Chokurdakh	River	Highly heterogeneous	8	±0.5	1.04	±0.06
Iultinsky (Chukotka)	River	Homogeneous	33	±0.5	1.32	±0.02
Iultinsky (Chukotka)	River	Heterogeneous	50	±0.5	2.0	±0.02
Iultinsky (Chukotka)	River	Highly heterogeneous	17	±0.5	0.68	±0.02
Southern Verkhoyansk Range	River	Homogeneous	33	±0.5	1.98	±0.03
Southern Verkhoyansk Range	River	Heterogeneous	28	±0.5	1.68	±0.03
Southern Verkhoyansk Range	River	Highly heterogeneous	39	±0.5	2.34	±0.03
Northern Olenek	Gully	Homogeneous	42	±0.5	1.68	±0.02
Northern Olenek	Gully	Heterogeneous	33	±0.5	1.32	±0.02
Northern Olenek	Gully	Highly heterogeneous	25	±0.5	1.0	±0.02
Chokurdakh	Gully	Highly homogeneous	61	±0.5	3.66	±0.03
Chokurdakh	Gully	Homogeneous	33	±0.5	1.98	±0.03
Chokurdakh	Gully	Heterogeneous	6	±0.5	0.36	±0.03
Iultinsky (Chukotka)	Gully	Homogeneous	28	±0.5	1.68	±0.03
Iultinsky (Chukotka)	Gully	Heterogeneous	56	±0.5	3.36	±0.03
Iultinsky (Chukotka)	Gully	Highly heterogeneous	17	±0.5	1.02	±0.03
Iultinsky (Chukotka)	Ponds + Gully	Heterogeneous	12	±0.5	1.32	±0.06
Iultinsky (Chukotka)	Ponds + Gully	Homogeneous	70	±0.5	7.7	±0.06
Iultinsky (Chukotka)	Ponds + Gully	Highly homogeneous	18	±0.5	1.98	±0.05
Southern Taymyr	River	-	0	±0.5	1.0	±0.0
Southern Verkhoyansk Range	Gully	-	0	±0.5	1.0	±0.0

Table 5: Table showing the rounding errors of the morphology related sunburst charts (Figure 23 to Figure 27). S.Percentage is the abbreviation of similarity percentage, which was rounded to 0 decimal places. This rounding results in a +/- 0.5% error. The Propagation of this error results in a section specific count error. The size of the individual sections of the sunburst chart was derived from Count. The count error describes the error in the size of each section, in other words, the error in the visualization. Importantly, the count errors apply to the visualisation only and do not influence the statistical analysis in this work.

Region	Morphology	Classification	S.Percentage [%]		S.Percentage error [%]	Count	Count error
Northern Olenek	Thermoterrace	Homogeneous	10	±0.5		1.0	±0.05
Northern Olenek	Thermoterrace	Heterogeneous	83	±0.5		8.3	±0.05
Northern Olenek	Thermoterrace	Highly heterogeneous	7	±0.5		0.7	±0.05
Chokurdakh	Thermoterrace	Homogeneous	48	±0.5		10.08	±0.1
Chokurdakh	Thermoterrace	Heterogeneous	52	±0.5		10.92	±0.11
Iultinsky (Chukotka)	Thermoterrace	Highly homogeneous	25	±0.5		1.0	±0.02
Iultinsky (Chukotka)	Thermoterrace	Homogeneous	50	±0.5		2.0	±0.02
Iultinsky (Chukotka)	Thermoterrace	Heterogeneous	25	±0.5		1.0	±0.02
Southern Taymyr	Combination	Homogeneous	39	±0.5		12.87	±0.16
Southern Taymyr	Combination	Heterogeneous	61	±0.5		20.13	±0.16
Northern Olenek	Combination	Highly homogeneous	3	±0.5		1.2	±0.2
Northern Olenek	Combination	Homogeneous	28	±0.5		11.2	±0.2
Northern Olenek	Combination	Heterogeneous	68	±0.5		27.2	±0.2
Chokurdakh	Combination	Homogeneous	29	±0.5		9.28	±0.16
Chokurdakh	Combination	Heterogeneous	70	±0.5		22.4	±0.16
Chokurdakh	Combination	Highly heterogeneous	1	±0.5		0.32	±0.16
Iultinsky (Chukotka)	Combination	Highly homogeneous	13	±0.5		4.55	±0.18
Iultinsky (Chukotka)	Combination	Homogeneous	70	±0.5		24.5	±0.18
Iultinsky (Chukotka)	Combination	Heterogeneous	17	±0.5		5.95	±0.18
Southern Verkhoyansk Range	Combination	Highly heterogeneous	100	±0.5		2.0	±0.01
Southern Taymyr	Thermocirque	Highly heterogeneous	2	±0.5		0.28	±0.07
Southern Taymyr	Thermocirque	Heterogeneous	33	±0.5		4.62	±0.07
Southern Taymyr	Thermocirque	Homogeneous	62	±0.5		8.68	±0.07
Southern Taymyr	Thermocirque	Highly homogeneous	2	±0.5		0.28	±0.07
Northern Olenek	Thermocirque	Highly heterogeneous	44	±0.5		1.32	±0.02
Northern Olenek	Thermocirque	Homogeneous	44	±0.5		1.32	±0.02
Northern Olenek	Thermocirque	Highly homogeneous	11	±0.5		0.33	±0.02
Chokurdakh	Thermocirque	Heterogeneous	17	±0.5		1.7	±0.05
Chokurdakh	Thermocirque	Homogeneous	57	±0.5		5.7	±0.05

Chokurdakh	Thermocirque	Highly homogeneous	27	±0.5	2.7	±0.05
Iultinsky (Chukotka)	Thermocirque	Heterogeneous	17	±0.5	3.74	±0.11
Iultinsky (Chukotka)	Thermocirque	Homogeneous	73	±0.5	16.06	±0.11
Iultinsky (Chukotka)	Thermocirque	Highly homogeneous	11	±0.5	2.42	±0.11
Southern Verkhoyansk Range	Thermocirque	Highly heterogeneous	11	±0.5	0.66	±0.03
Southern Verkhoyansk Range	Thermocirque	Heterogeneous	67	±0.5	4.02	±0.03
Southern Verkhoyansk Range	Thermocirque	Homogeneous	22	±0.5	1.32	±0.03
Southern Verkhoyansk Range	Thermoterrace	-	0	±0.5	1.0	±0.0

Appendix C6: Differences in the generosity of mapped RTS outlines.

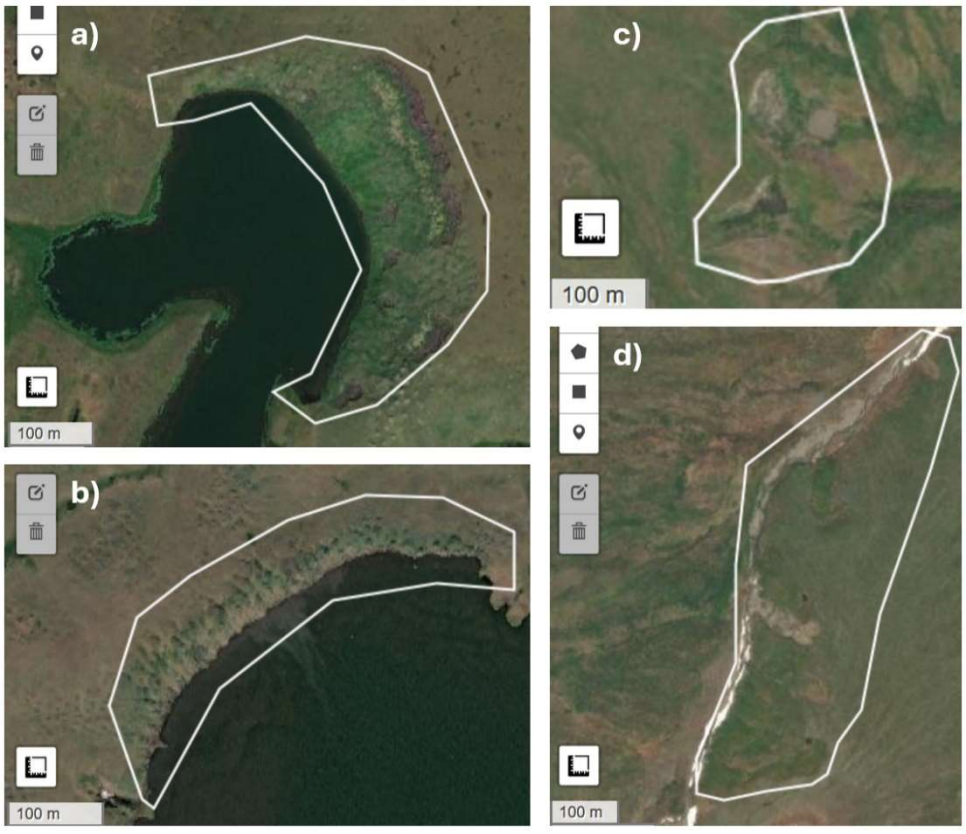


Figure 34: ***Differences in the generosity of mapped RTS outlines.*** *The figure shows variations in how RTS outlines were drawn. More generous outlines* ***(c, d)*** *include additional surrounding terrain, while stricter outlines* ***(a, b)*** *result in more confined RTS areas.*

Appendix D: Analysis Code

```python
# -*- coding: utf-8 -*-

# Insert GEE
"""

import ee
import geemap

geemap.ee_initialize()

"""# Loading ALEX"""

# Load the asset
alex = ee.ImageCollection('users/ingmarnitze/TCTrend_SR_2003-2022_TCVIS') #Every image (of the 353 images) shows a different area
alex

"""# Loading Validation Dataset 2"""

rtsT1 =  ee.FeatureCollection('projects/ee-moritzjulia7/assets/RTS_T1')
rtsT2 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/RTS_T2')
rtsT3 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/RTS_T3')
rtsT4_all = ee.FeatureCollection('projects/ee-moritzjulia7/assets/RTS_T4_all')
rtsT6_all = ee.FeatureCollection('projects/ee-moritzjulia7/assets/RTS_T6_all')

"""Extract uncertain rts from T4 and T6"""

rtsT4 = rtsT4_all.filter(ee.Filter.eq('Uncertain', 0))
rtsT6 = rtsT6_all.filter(ee.Filter.eq('Uncertain', 0))

area1 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/Area_T1')
area2 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/Area_T2')
area3 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/Area_T3')
area4 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/Area_T4')
area6 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/Area_T6')

SubAreas1 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/SubAreas_T1')
SubAreas2 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/SubAreas_T2')
SubAreas3 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/SubAreas_T3')
SubAreas4 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/SubAreas_T4')
SubAreas6 = ee.FeatureCollection('projects/ee-moritzjulia7/assets/SubAreas_T6')

"""What image of ALEX shows which study area?"""

image_index = 174  # Change this to the desired index (0 for first, 1 for second, 2 for third, etc.)
image = alex.toList(alex.size()).get(image_index)
image = ee.Image(image)

Map=geemap.Map()

style = {
    'color': 'red',      # Outline color
    'fillColor': '00000000', # Transparent fill
    'width': 2
}

Map.addLayer(image, {}, 'ALEX')
Map.addLayer(area1.style(**style), {}, "Area_T1")
Map.addLayer(area2.style(**style), {}, "Area_T2")
Map.addLayer(area3.style(**style), {}, "Area_T3")
Map.addLayer(area4.style(**style), {}, "Area_T4")
Map.addLayer(area6.style(**style), {}, "Area_T6")
Map.setCenter(98.39,72.71,3)
Map

## Results:
# Area 1: 116
# Area 2: 140
# Area 3: 174
# Area 4: 351
# Area 6: 161
```

```
71
72  """Creating ALEX subset for each region"""
73
74  #Area 1
75  image_index = 116
76  image1 = alex.toList(alex.size()).get(image_index)
77  alex_area1 = ee.Image(image1)
78
79  #Area 2
80  image_index = 140
81  image2 = alex.toList(alex.size()).get(image_index)
82  alex_area2 = ee.Image(image2)
83
84  #Area 3
85  image_index = 174
86  image3 = alex.toList(alex.size()).get(image_index)
87  alex_area3 = ee.Image(image3)
88
89  #Area 4
90  image_index = 351
91  image4 = alex.toList(alex.size()).get(image_index)
92  alex_area4 = ee.Image(image4)
93
94  #Area 6
95  image_index = 161
96  image6 = alex.toList(alex.size()).get(image_index)
97  alex_area6 = ee.Image(image6)
98
99  """# Data Preparation per Region (sub-hypothesis 1)
100
101 Creating ImageCollection of ALEX data within RTS polygons per Area
102 """
103
104 ## Area 1
105
106 # Empty list to store the features
107 rtsT1_alex_features = []
108
109 # Get the number of features in the rtsT1 feature collection
110 rtsT1_size = rtsT1.size().getInfo()
111
112 # Iterate through each feature in rtsT1
113 for i in range(rtsT1_size):
114   # Get the current feature
115   rtsT1_feature = ee.Feature(rtsT1.toList(rtsT1_size).get(i))
116
117   # Clip alex_area1 to the current feature's geometry
118   single_rtsT1_alex_feature = alex_area1.clip(rtsT1_feature.geometry())
119
120   # Append the feature to the list
121   rtsT1_alex_features.append(single_rtsT1_alex_feature)
122
123 # Convert the list of features to a ImageCollection
124 rtsT1_alex_imagecollection = ee.ImageCollection(rtsT1_alex_features)
125
126
127 # Display the ImageCollection on a map
128 Map = geemap.Map()
129 Map.addLayer(rtsT1_alex_imagecollection, {}, 'ALEX Images')
130 Map.setCenter(98.39,72.71,12)
131 Map
132
133 ## Area 2
134
135 rtsT2_alex_features = []
136
137 rtsT2_size = rtsT2.size().getInfo()
138
139 for i in range(rtsT2_size):
140   rtsT2_feature = ee.Feature(rtsT2.toList(rtsT2_size).get(i))
141
142   single_rtsT2_alex_feature = alex_area2.clip(rtsT2_feature.geometry())
```

```
    rtsT2_alex_features.append(single_rtsT2_alex_feature)

rtsT2_alex_imagecollection = ee.ImageCollection(rtsT2_alex_features)
rtsT2_alex_imagecollection

## Area 3

rtsT3_alex_features = []

rtsT3_size = rtsT3.size().getInfo()

for i in range(rtsT3_size):
  rtsT3_feature = ee.Feature(rtsT3.toList(rtsT3_size).get(i))

  single_rtsT3_alex_feature = alex_area3.clip(rtsT3_feature.geometry())

  rtsT3_alex_features.append(single_rtsT3_alex_feature)

rtsT3_alex_imagecollection = ee.ImageCollection(rtsT3_alex_features)
rtsT3_alex_imagecollection

## Area 4

rtsT4_alex_features = []

rtsT4_size = rtsT4.size().getInfo()

for i in range(rtsT4_size):
  rtsT4_feature = ee.Feature(rtsT4.toList(rtsT4_size).get(i))

  single_rtsT4_alex_feature = alex_area4.clip(rtsT4_feature.geometry())

  rtsT4_alex_features.append(single_rtsT4_alex_feature)

rtsT4_alex_imagecollection = ee.ImageCollection(rtsT4_alex_features)
rtsT4_alex_imagecollection

## Area 6

rtsT6_alex_features = []

rtsT6_size = rtsT6.size().getInfo()

for i in range(rtsT6_size):
  rtsT6_feature = ee.Feature(rtsT6.toList(rtsT6_size).get(i))

  single_rtsT6_alex_feature = alex_area6.clip(rtsT6_feature.geometry())

  rtsT6_alex_features.append(single_rtsT6_alex_feature)

rtsT6_alex_imagecollection = ee.ImageCollection(rtsT6_alex_features)
rtsT6_alex_imagecollection

"""Funktion to build the GeoDataframes"""

import geopandas as gpd
import pandas as pd
from shapely.geometry import Point, Polygon
from shapely.geometry import shape

band_names = ['TCW_slope', 'TCB_slope', 'TCG_slope']

def image_to_geodataframe(image, bands, scale=30):
    # Sample the image to get data as a FeatureCollection
    fc = image.select(bands).sample(scale=scale, geometries=True)
    geojson = fc.getInfo()

    features = geojson['features']
    rows = []
    for feature in features:
        properties = feature['properties']
```

```
            coords = feature['geometry']['coordinates']
            point_geometry = Point(coords)
            rows.append({
                **properties,
                'geometry': point_geometry
            })

        gdf = gpd.GeoDataFrame(rows, crs="EPSG:4326")
        return gdf

    # Create GeoDataFrames for each image in the ImageCollection
    def collection_to_geodataframes(image_collection, bands, scale=30):
        images = image_collection.toList(image_collection.size())
        num_images = image_collection.size().getInfo()

        geodataframes = []
        for i in range(num_images):
            image = ee.Image(images.get(i))
            gdf = image_to_geodataframe(image, bands, scale)
            geodataframes.append(gdf)

        return geodataframes

    """Building DataFrame per single RTS for each study Area"""

    ## Area 1
    band_names = ['TCW_slope', 'TCB_slope', 'TCG_slope']
    gdfs_T1 = collection_to_geodataframes(rtsT1_alex_imagecollection, band_names)

    print(gdfs_T1[0])

    ## Area 2
    gdfs_T2 = collection_to_geodataframes(rtsT2_alex_imagecollection, band_names)

    print(gdfs_T2[0])

    ## Area 3
    gdfs_T3 = collection_to_geodataframes(rtsT3_alex_imagecollection, band_names)

    print(gdfs_T3[0])

    ## Area 4
    gdfs_T4 = collection_to_geodataframes(rtsT4_alex_imagecollection, band_names)

    print(gdfs_T4[0])

    ## Area 6
    gdfs_T6 = collection_to_geodataframes(rtsT6_alex_imagecollection, band_names)

    print(gdfs_T6[0])

    """Create one Dataframe per Area containig all values"""

    gdf_T1 = gpd.GeoDataFrame(pd.concat(gdfs_T1, ignore_index=True), crs=gdfs_T1[0].crs)
    gdf_T2 = gpd.GeoDataFrame(pd.concat(gdfs_T2, ignore_index=True), crs=gdfs_T2[0].crs)
    gdf_T3 = gpd.GeoDataFrame(pd.concat(gdfs_T3, ignore_index=True), crs=gdfs_T3[0].crs)
    gdf_T4 = gpd.GeoDataFrame(pd.concat(gdfs_T4, ignore_index=True), crs=gdfs_T4[0].crs)
    gdf_T6 = gpd.GeoDataFrame(pd.concat(gdfs_T6, ignore_index=True), crs=gdfs_T6[0].crs)

    print(gdf_T1)

    """Saving Dataframes with all data of RTSs per area (merged)"""

    from google.colab import drive

    #drive.mount('/content/drive')
    #gdf_T1.to_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T1.shp' )
    #gdf_T2.to_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T2.shp' )
    #gdf_T3.to_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T3.shp' )
    #gdf_T4.to_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T4.shp' )
    #gdf_T6.to_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T6.shp' )
```

```
"""Load data frames with all data of RTSs per area (merged)"""

import geopandas as gpd
from google.colab import drive

drive.mount('/content/drive')
gdf_T1 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T1.shp' )
gdf_T2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T2.shp' )
gdf_T3 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T3.shp' )
gdf_T4 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T4.shp' )
gdf_T6 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/gdf_T6.shp' )

"""Dataframes rescaling (normalisation)"""

gdf_T1_norm = gdf_T1[['TCB_slope', 'TCG_slope', 'TCW_slope']] / 255. *0.24 - 0.12
gdf_T1_norm['geometry'] = gdf_T1['geometry']
gdf_T2_norm = gdf_T2[['TCB_slope', 'TCG_slope', 'TCW_slope']] / 255. *0.24 - 0.12
gdf_T2_norm['geometry'] = gdf_T2['geometry']
gdf_T3_norm = gdf_T3[['TCB_slope', 'TCG_slope', 'TCW_slope']] / 255. *0.24 - 0.12
gdf_T3_norm['geometry'] = gdf_T3['geometry']
gdf_T4_norm = gdf_T4[['TCB_slope', 'TCG_slope', 'TCW_slope']] / 255. *0.24 - 0.12
gdf_T4_norm['geometry'] = gdf_T4['geometry']
gdf_T6_norm = gdf_T6[['TCB_slope', 'TCG_slope', 'TCW_slope']] / 255. *0.24 - 0.12
gdf_T6_norm['geometry'] = gdf_T6['geometry']

"""## Histogramms of areas"""

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

#single Historgrams for overview map
colors = {
    "TCB_slope": "#FF3333",  # Middle red intense
    "TCG_slope": "#00FF00",   # Bright green intense
    "TCW_slope": "#003366"    # Middle blue intense
}

# Create the plot
plt.figure(figsize=(10, 6))

for column, color in colors.items():
    sns.histplot(data=gdf_T6_norm, x=column, color=color, kde=True, label=column, bins
    =30)

plt.axvline(0, color='black', linestyle='--', linewidth=1)
plt.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.7)

# Adjust the y-axis to represent area
ax = plt.gca()  # Get the current Axes
y_ticks = ax.get_yticks()  # Get current y-tick positions
ax.set_yticklabels([f"{int(y * 900):,}" for y in y_ticks])  # Convert frequency to
area (900 m² per count)


# Add labels and title
plt.xlabel("Slope Value")
plt.ylabel("Area (m²)")
plt.title("Histogram of the Spectral Slope of RTSs Covering the Region of Southern
Verkhoyansk Range, Siberia")
plt.legend(title="Slope Type")

#plt.savefig("Hist_Area6_SouthernVerkhoyanskRange.svg", format="svg")
plt.show()

# all Histograms on same y-axis
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
titles = ["Southern Taymyr", "Northern Olenek", "Chokurdakh", "Iultinsky (Chukotka)",
"Southern Verkhoyansk Range"] #["Area 1", "Area 2", "Area 3", "Area 4", "Area 6"]

colors = {
    "TCB_slope": "#FF3333",
```

```
    "TCG_slope": "#00FF00",
    "TCW_slope": "#003366"
    }

# Create a figure
fig, axes = plt.subplots(1, 5, figsize=(20, 5), sharey=True)

# Iterate over datasets and plot each histogram in a separate subplot
for i, (ax, df, title) in enumerate(zip(axes, dataframes, titles)):
    for column, color in colors.items():
        sns.histplot(data=df, x=column, color=color, kde=True, label=column, bins=30,
        ax=ax)
        ax.axvline(0, color='black', linestyle='--', linewidth=1)
        ax.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.7)
    ax.set_title(title)
    ax.set_xlabel("Slope Value")
    if i == 0:  # Add Y-axis label to the first subplot only
        ax.set_ylabel("Area (m²)")
    else:
        ax.set_ylabel("")

ax = plt.gca()  # Get the current Axes
y_ticks = ax.get_yticks()  # Get current y-tick positions
ax.set_yticklabels([f"{int(y * 900):,}" for y in y_ticks])  # Convert frequency to
area (900 m² per count)

fig.suptitle("Histograms of the Spectral Slope of RTSs Covering Different Regions of
Siberia", fontsize=16)

axes[0].legend(title="Slope Type", loc='upper right')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])
#plt.savefig("Hist_All_Areas_RegionNames.svg", format="svg")
plt.show()

"""## Statistical tests Sub-Hyp-1"""

from scipy.stats import shapiro
from scipy.stats import anderson

"""Testing normal distribution"""

## shapiro - test
# List of dataframes
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]

# Columns to test
columns = ["TCB_slope", "TCG_slope", "TCW_slope"]

# Store the results
shapiro_results = {}

for i, df in enumerate(dataframes):
    df_name = f"gdf_T{i+1}_norm"  # Name for each dataframe
    shapiro_results[df_name] = {}

    for col in columns:
        stat, p_value = shapiro(df[col])
        shapiro_results[df_name][col] = {"statistic": stat, "p_value": p_value}

# Print the results
for df_name, results in shapiro_results.items():
    print(f"\nShapiro-Wilk Test Results for {df_name}:")
    for col, result in results.items():
        print(f"  {col}: statistic={result['statistic']:.4f}, p-value={result[
        'p_value']:.4f}")

##Anderson-Darling test
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]
```

```
# Loop through each DataFrame and each column
for i, df in enumerate(dataframes):
    print(f"\nDataFrame {i+1}:")
    for column in columns_to_test:
        print(f"Testing column: {column}")
        # Perform the Anderson-Darling test
        result = anderson(df[column], dist="norm")

        # Print the results
        print(f"  Statistic: {result.statistic:.4f}")
        print(f"  Critical Values:")
        for level, critical_value in zip(result.significance_level, result.
        critical_values):
            print(f"    {level}%: {critical_value:.4f}")

        # Interpret results (using 5% significance level as an example)
        if result.statistic > result.critical_values[2]:  # 5% level
            print(f"  Result: The data does NOT follow a normal distribution (reject
            H0).")
        else:
            print(f"  Result: The data appears to follow a normal distribution (fail
            to reject H0).")

"""Using Kruskal-Wallis test to ceck if all medians for each group (TCW, TCB, TCG) of
all areas are equal"""

from scipy.stats import kruskal

## Kruskal-Wallis H-test
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]

# Loop through each column and perform Kruskal-Wallis test
for column in columns_to_test:
    print(f"\nTesting column: {column}")

    # Extract the data for the current column from each DataFrame
    data = [df[column].dropna() for df in dataframes]

    # Perform the Kruskal-Wallis H-test
    statistic, p_value = kruskal(*data)

    # Print the results
    print(f"  H-statistic: {statistic:.4f}")
    print(f"  p-value: {p_value:.4f}")

    # Interpretation based on a 5% significance level
    if p_value < 0.05:
        print("  Result: There is a significant difference between the sample sites
        (reject H0).")
    else:
        print("  Result: No significant difference between the sample sites (fail to
        reject H0).")

"""Dunn's Test for Pairwise Comparisons"""

!pip install scikit-posthocs
# during analysis the version "0.11.2-py3-none-any.whl.metadata (5.8 kB)" was used.
#This version is no longer availabe, version 0.11.3 can produce slightly different
results.
import scikit_posthocs as sp
import pandas as pd

#Dunn's test
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]
sample_labels = ['T1', 'T2', 'T3', 'T4', 'T6']

# Loop through each column and perform Dunn's test
for column in columns_to_test:
    print(f"\nTesting column: {column}")

```

```
        # Create a list of the values for the column from each DataFrame
        data = [df[column].dropna() for df in dataframes]

        # Create a list of group labels
        labels = []
        for i, df in enumerate(dataframes):
            labels.extend([sample_labels[i]] * len(df[column].dropna()))

        # Perform Dunn's test (pairwise comparisons)
        p_values = sp.posthoc_dunn(data, p_adjust="bonferroni")  # Using Bonferroni correction
        print(p_values)

        # Interpret results
        print("Pairwise comparisons (p-values) with Bonferroni correction:")
        for i in range(len(p_values.columns)):
            for j in range(i+1, len(p_values.columns)):
                p_val = p_values.iloc[i, j]
                if p_val < 0.05:
                    print(f"  Significant difference between {sample_labels[i]} and {sample_labels[j]}: p = {p_val:.4f}")
                else:
                    print(f"  No significant difference between {sample_labels[i]} and {sample_labels[j]}: p = {p_val:.4f}")

# Collecting dunns test results in df
dataframes = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]
sample_labels = ["Southern Taymyr", "Northern Olenek", "Chokurdakh", "Iultinsky (C.)", "S. Verkhoyansk Range"] #['T1', 'T2', 'T3', 'T4', 'T6']


p_values_Dunn = {}

for column in columns_to_test:
        # Create a list of the values for the column from each DataFrame
    data = [df[column].dropna() for df in dataframes]

    # Perform Dunn's test (pairwise comparisons)
    p_values = sp.posthoc_dunn(data, p_adjust="bonferroni")  # Using Bonferroni correction

    p_values_Dunn[column] = pd.DataFrame(
        p_values.values,
        columns=sample_labels,
        index=sample_labels
    )
p_values_Dunn

"""Visualize Statistics"""

import matplotlib.pyplot as plt
import seaborn as sns
# older verion (v0.13.2) was used, new version is v0.14.0
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap, Normalize

# Custom colormap with transitions starting at 0, 0.05 and 1
colors = [
    (0.5, 0.1, 0.5),
    (1, 0.8, 0.4),        # Yellow starts at 0.05
    (1, 0.8, 0.4)
]
positions = [0.0000, 0.0500, 1]  # Color start/end point
cmap_name = "custom_gradient_cmap"
smooth_cmap = LinearSegmentedColormap.from_list(cmap_name, list(zip(positions, colors)))

# Normalize to align colors with specific ranges
norm = Normalize(vmin=0, vmax=1)

```

```
# Number of subplots (columns)
num_columns = len(p_values_Dunn)
fig, axes = plt.subplots(num_columns, 1, figsize=(7, 6 * num_columns), sharey=True)

# Loop through the dictionary and plot each matrix
for ax, (column, matrix) in zip(axes, p_values_Dunn.items()):
    sns.heatmap(
        matrix,
        annot=True,
        fmt=".4f",
        cmap=smooth_cmap,
        norm=norm,
        cbar_kws={'label': 'p-value'},
        ax=ax
    )
    ax.set_title(f'Dunn-Bonferroni-Test P-Values: {column}')
    ax.set_xlabel('Areas')
    ax.set_ylabel('Areas')

plt.tight_layout()
plt.show()

"""Violine Plots - Visualization of Areas"""

# Combine data for all areas and slopes
df_combined = pd.concat([
    gdf_T1_norm[['TCB_slope', 'TCW_slope', 'TCG_slope']].assign(site='Southern Taymyr'),
    gdf_T2_norm[['TCB_slope', 'TCW_slope', 'TCG_slope']].assign(site='Northern Olenek'),
    gdf_T3_norm[['TCB_slope', 'TCW_slope', 'TCG_slope']].assign(site='Chokurdakh'),
    gdf_T4_norm[['TCB_slope', 'TCW_slope', 'TCG_slope']].assign(site='Iultinsky'),
    gdf_T6_norm[['TCB_slope', 'TCW_slope', 'TCG_slope']].assign(site='S. Verkhoyansk Range')
])

# Melt the data for plotting
df_melted = df_combined.melt(id_vars=['site'], value_vars=['TCB_slope', 'TCW_slope', 'TCG_slope'],
                             var_name='Slope Type', value_name='Slope Value')

# Set the palette for the plot
palette = sns.color_palette("Set2")

# Create the violin plot for all bands
plt.figure(figsize=(15, 6))
sns.violinplot(x="Slope Type", y="Slope Value", hue="site", data=df_melted, palette=palette, dodge=True)

# Customize the plot
plt.title('Violin Plots of the Slopes (TCB, TCW, TCG) for All Areas')
plt.ylabel('Slope Value')
plt.xlabel('Slope Type')
plt.axhline(y=0, color='dimgrey', linestyle='--', zorder=1)
plt.legend(title="Areas", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

"""# Data collection for latitudes and longitudes plots"""

import numpy as np
from shapely.geometry import Point
import pandas as pd

AreaCollection = [gdf_T1_norm, gdf_T2_norm, gdf_T3_norm, gdf_T4_norm, gdf_T6_norm]
AreaCollection_names = ['Southern Taymyr', 'Northern Olenek', 'Chokurdakh', 'Iultinsky (C.)', 'S. Verkhoyansk Range']

# Create an empty list to store the data for each area
data = []

for ind in np.arange(len(AreaCollection)):
```

```
        temp_df = AreaCollection[ind]

        # Calculate the center coordinates
        mean_coordinates = np.mean(np.array([[geom.x, geom.y] for geom in temp_df.geometry
        ]), axis=0)
        mean_coordinates = np.abs(mean_coordinates)

        # Calculate the mean values for each channel
        meanVals = np.zeros(3)
        meanVals[0] = np.mean(temp_df['TCB_slope'])
        meanVals[1] = np.mean(temp_df['TCG_slope'])
        meanVals[2] = np.mean(temp_df['TCW_slope'])

        # Append the data for the current area to the list
        data.append([
            AreaCollection_names[ind],  # Area name
            mean_coordinates[0],        # Center X coordinate
            mean_coordinates[1],        # Center Y coordinate
            Point(mean_coordinates),    # Center point as a Point object
            meanVals[0],                # Mean TCB value
            meanVals[1],                # Mean TCG value
            meanVals[2]                 # Mean TCW value
        ])

    # Create a Pandas DataFrame from the data list
    AreaOverview_df = pd.DataFrame(data, columns=[
        'name', 'center_x', 'center_y', 'center_point',
        'TCB_norm_mean', 'TCG_norm_mean', 'TCW_norm_mean'
    ])

    print(AreaOverview_df)

    """## Longitude Latitude Plots"""

    import matplotlib.pyplot as plt
    from matplotlib.lines import Line2D
    import plotly.express as px
    import pandas as pd
    import numpy as np
    import statsmodels.api as sm

    def plot_regression_with_ci(x, y, color, label):

        # Sort the data by x for consistent plotting
        sort_idx = np.argsort(x)
        x_sorted = x[sort_idx]
        y_sorted = y[sort_idx]

        # Fit the regression model
        X = sm.add_constant(x_sorted)  # Add constant for the intercept
        model = sm.OLS(y_sorted, X).fit()  # Ordinary Least Squares regression
        predictions = model.predict(X)  # Predicted values

        # Get confidence intervals
        prediction_summary = model.get_prediction(X).summary_frame(alpha=0.05)  # 95% CI
        ci_lower = prediction_summary["mean_ci_lower"]
        ci_upper = prediction_summary["mean_ci_upper"]

        # Plot the regression line
        plt.plot(x_sorted, predictions, color=color, linestyle="--", label=f"{label}
        Regression")

        # Plot the confidence interval
        plt.fill_between(x_sorted, ci_lower, ci_upper, color=color, alpha=0.2, label=f"{
        label} CI")

    plt.figure(figsize=(10, 6))
    plt.scatter(AreaOverview_df["center_y"], AreaOverview_df["TCB_norm_mean"], color="red"
    , label="TCB Mean", marker="x", linestyle="None")
    plt.scatter(AreaOverview_df["center_y"], AreaOverview_df["TCG_norm_mean"], color=
    "green", label="TCG Mean", marker="x", linestyle="None")
    plt.scatter(AreaOverview_df["center_y"], AreaOverview_df["TCW_norm_mean"], color=
```

```
    "blue", label="TCW Mean", marker="x", linestyle="None")
687 plot_regression_with_ci(AreaOverview_df["center_y"], AreaOverview_df["TCB_norm_mean"],
     "red", "TCB")
688 plot_regression_with_ci(AreaOverview_df["center_y"], AreaOverview_df["TCG_norm_mean"],
     "green", "TCG")
689 plot_regression_with_ci(AreaOverview_df["center_y"], AreaOverview_df["TCW_norm_mean"],
     "blue", "TCW")
690
691
692 for i, area_name in enumerate(AreaOverview_df["name"]):
693
694     plt.text(
695         AreaOverview_df["center_y"][i],
696         0.026,  # Adjust annotation height for clarity
697         area_name,
698         fontsize=9,
699         ha="center",
700         rotation=90
701     )
702
703 # Finalize the plot
704 plt.title("Mean of Slope per Area vs. Latitude")
705 plt.xlabel("Latitude")
706 plt.ylabel("Mean of Slope")
707 plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")  # Legend outside the plot
708 plt.tight_layout()
709 plt.show()
710
711 plt.figure(figsize=(10, 6))
712 plt.scatter(AreaOverview_df["center_x"], AreaOverview_df["TCB_norm_mean"], color="red"
    , label="TCB Mean", marker="x", linestyle="None")
713 plt.scatter(AreaOverview_df["center_x"], AreaOverview_df["TCG_norm_mean"], color=
    "green", label="TCG Mean", marker="x", linestyle="None")
714 plt.scatter(AreaOverview_df["center_x"], AreaOverview_df["TCW_norm_mean"], color=
    "blue", label="TCW Mean", marker="x", linestyle="None")
715 plot_regression_with_ci(AreaOverview_df["center_x"], AreaOverview_df["TCB_norm_mean"],
     "red", "TCB")
716 plot_regression_with_ci(AreaOverview_df["center_x"], AreaOverview_df["TCG_norm_mean"],
     "green", "TCG")
717 plot_regression_with_ci(AreaOverview_df["center_x"], AreaOverview_df["TCW_norm_mean"],
     "blue", "TCW")
718
719
720 for i, area_name in enumerate(AreaOverview_df["name"]):
721
722     plt.text(
723         AreaOverview_df["center_x"][i],
724         0.027,  # Adjust annotation height for clarity
725         area_name,
726         fontsize=9,
727         ha="center",
728         rotation=90
729     )
730
731 # Finalize the plot
732 plt.title("Mean of Slope per Area vs. Longitude")
733 plt.xlabel("Longitude")
734 plt.ylabel("Mean of Slope")
735 plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")  # Legend outside the plot
736 plt.tight_layout()
737 plt.show()
738
739 """# Sub-Areas (sub-hypothsis 2)"""
740
741 import geopandas as gpd
742 from shapely.geometry import shape
743 from shapely.geometry import Point
744
745 """## Loop to generate dicts per Area containing sub area gdfs"""
746
747 subareas = {
748     "SubAreas1": SubAreas1,
```

```
    "SubAreas2": SubAreas2,
    "SubAreas3": SubAreas3,
    "SubAreas4": SubAreas4,
    "SubAreas6": SubAreas6
}

gdfs = {
    "gdf_T1_norm": gdf_T1_norm,
    "gdf_T2_norm": gdf_T2_norm,
    "gdf_T3_norm": gdf_T3_norm,
    "gdf_T4_norm": gdf_T4_norm,
    "gdf_T6_norm": gdf_T6_norm
}

all_gdf_SubAreas = {}

# Loop through each subarea-feature-collection pair
for subarea_name, subarea_fc in subareas.items():
    # Get the corresponding GeoDataFrame for the subarea
    gdf_name = f"gdf_T{subarea_name[-1]}_norm"
    gdf = gdfs[gdf_name]

    gdf_dict = {}

    # Iterate through the features (squares) in the current subarea
    for i, square_feature in enumerate(subarea_fc.getInfo()['features']):
        square_geometry = shape(square_feature['geometry'])

        filtered_points = gdf[gdf.geometry.apply(lambda geom: geom.within(
        square_geometry))]

        if filtered_points.empty:
            print(f"Warning: No points found in {subarea_name} square {i + 1}.
            Skipping this square.")
            continue

        gdf_dict[f"{gdf_name}_{i + 1}"] = filtered_points

    all_gdf_SubAreas[subarea_name] = gdf_dict

    # Print keys of the current dictionary for verification
    print(f"Created GeoDataFrames for {subarea_name}:", list(gdf_dict.keys()))

"""## Loop to calculate statistics for all sub areas of all areas"""

from scipy.stats import shapiro
from scipy.stats import anderson
from scipy.stats import kruskal
!pip install scikit-posthocs
# during analysis the version "0.11.2-py3-none-any.whl.metadata (5.8 kB)" was used.
#Version 0.11.2 is no longer availabe, version 0.11.3 can produce slightly different
results.
import scikit_posthocs as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap, BoundaryNorm, Normalize

# shapiro
columns = ["TCB_slope", "TCG_slope", "TCW_slope"]

# Store the results
shapiro_results = {}

# Iterate through all SubAreas in all_gdf_SubAreas
for subarea_name, subarea_data in all_gdf_SubAreas.items():
    shapiro_results[subarea_name] = {}

    for df_name, df in subarea_data.items():
        shapiro_results[subarea_name][df_name] = {}

```

```
            for col in columns:
                try:
                    # Perform Shapiro-Wilk test
                    stat, p_value = shapiro(df[col])
                    shapiro_results[subarea_name][df_name][col] = {"statistic": stat, "p_value": p_value}
                except Exception as e:
                    # Handle cases where the test cannot be performed
                    shapiro_results[subarea_name][df_name][col] = {"statistic": None, "p_value": None, "error": str(e)}

    # Print the results
    for subarea_name, subarea_results in shapiro_results.items():
        print(f"\nShapiro-Wilk Test Results for {subarea_name}:")
        for df_name, results in subarea_results.items():
            print(f"  DataFrame: {df_name}")
            for col, result in results.items():
                if result["statistic"] is not None:
                    print(f"    {col}: statistic={result['statistic']:.4f}, p-value={result['p_value']:.4f}")
                else:
                    print(f"    {col}: Test could not be performed. Error: {result['error']}")

    # Kruskal

    columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]

    # Loop through each SubArea in all_gdf_SubAreas
    for subarea_key, subarea_df_dict in all_gdf_SubAreas.items():
        print(f"\nTesting for {subarea_key}")

        # Loop through each column and perform Kruskal-Wallis test
        for column in columns_to_test:
            print(f"\n  Testing column: {column}")

            # Extract the data for the current column from each DataFrame in the current SubArea
            data = [df[column].dropna() for df_name, df in subarea_df_dict.items()]

            # Perform the Kruskal-Wallis H-test
            statistic, p_value = kruskal(*data)

            # Print the results
            print(f"    H-statistic: {statistic:.4f}")
            print(f"    p-value: {p_value:.4f}")

            # Interpretation based on a 5% significance level
            if p_value < 0.05:
                print("    Result: There is a significant difference between the sample sites (reject H0).")
            else:
                print("    Result: No significant difference between the sample sites (fail to reject H0).")

    # Dunn
    columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]

    p_values_Dunn = {}

    # Loop through each SubArea in all_gdf_SubAreas
    for subarea_key, subarea_df_dict in all_gdf_SubAreas.items():
        print(f"\nPerforming Dunn's Test for {subarea_key}")

        p_values_Dunn[subarea_key] = {}

        # Loop through each column and perform Dunn's test
        for column in columns_to_test:
            print(f"  Testing column: {column}")

            # Create a list of the values for the column from each DataFrame in the current SubArea
```

```
882             data = [df[column].dropna() for df_name, df in subarea_df_dict.items()]
883
884             sample_labels = [df_name.split('_')[-1] for df_name in subarea_df_dict.keys()]
885
886             # Perform Dunn's test (pairwise comparisons)
887             p_values = sp.posthoc_dunn(data, p_adjust="bonferroni")  # Using Bonferroni
                correction
888
889             # Store the p-values as a DataFrame for the current column and SubArea
890             p_values_Dunn[subarea_key][column] = pd.DataFrame(
891                 p_values.values,
892                 columns=sample_labels,
893                 index=sample_labels
894             )
895
896     # To check the results for each SubArea and column, printing the p_values_Dunn
        dictionary
897     p_values_Dunn
898
899     # Custom colormap
900       colors = [
901           (0.5, 0.1, 0.5),
902           (1, 0.8, 0.4),      # Yellow starts at 0.05
903           (1, 0.8, 0.4)
904       ]
905       positions = [0.0000, 0.0500, 1]  # Define where each color starts/ends
906       cmap_name = "custom_gradient_cmap"
907       smooth_cmap = LinearSegmentedColormap.from_list(cmap_name, list(zip(positions,
          colors)))
908
909       # Normalize to align colors with specific ranges
910       norm = Normalize(vmin=0, vmax=1)
911
912       # Create a subplot grid with 1 row and number of columns based on the number of
          SubAreas
913       fig, axes = plt.subplots(5, 3, figsize=(20,20))
914
915       axes_flat = axes.flatten()
916
917       # Loop through the p_values_Dunn dictionary and plot each matrix
918       k = 0
919       for i, (subarea_key, subarea_p_values) in enumerate(p_values_Dunn.items()):
920           for j, (column, matrix) in enumerate(subarea_p_values.items()):
921               ax = axes_flat[k]
922               k += 1
923
924               # Plot the heatmap for the current subarea and column
925               sample_labels = list(matrix.columns)
926
927               # Plot heatmap for each SubArea and column
928               sns.heatmap(
929                   matrix,
930                   annot=True,
931                   fmt=".4f",
932                   cmap=smooth_cmap,
933                   norm=norm,
934                   cbar_kws={'label': 'p-value'},
935                   ax=ax
936               )
937
938               ax.set_title(f'Dunn-Bonferroni-Test P-Values: {subarea_key} - {column}')
939               ax.set_xlabel('Sub Areas')
940               ax.set_ylabel('Sub Areas')
941
942               # Get the current x-tick locations
943               xticks = ax.get_xticks()
944
945               # Set x-tick labels only for the available tick locations
946               ax.set_xticks(xticks)
947               ax.set_xticklabels(sample_labels[:len(xticks)], rotation=0, ha='right')
948
949               # Similarly, for y-axis:
```

```
            yticks = ax.get_yticks()
            ax.set_yticks(yticks)
            ax.set_yticklabels(sample_labels[:len(yticks)], rotation=0)
    plt.tight_layout()

    #plt.savefig(r'C:\Users\morit\Documents\Geoökologie\Module\BachelorArbeit\Daten\Plots\Dunn-Bonnferroni-Test_Sub-Areas.svg')
    #plt.savefig(f'/content/drive/My Drive/Colab Notebooks/Data/Dunn-Bonnferroni-Test_Sub-Areas.svg', format='svg')
    plt.show()

"""## Classification of Dunn values (homogeneity/heterogeneity)"""

import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.table import Table
import matplotlib.patches as patches

# df that contains information on sub areas exceeding 0.05 (-> are similar)
data = p_values_Dunn

# Initialize an empty list to store the results
results = []

# Iterate through the subareas and slope types
for subarea_key, slopes in data.items():
    for slope_key, matrix in slopes.items():
        # Compute row counts exceeding 0.05
        for row_index, row_values in matrix.iterrows():
            total_columns = len(row_values) -1
            exceed_count = (row_values > 0.05).sum() -1
            percentage = (exceed_count / total_columns) * 100
            modified_subarea_key = subarea_key[3:]  # Remove first three character
            modified_subarea_key = modified_subarea_key[:4] + modified_subarea_key[5:]
             # Remove 8. character
            results.append({
                "Area": modified_subarea_key,
                "Slope": slope_key,
                "SubAreaIndex": row_index,
                "TotalColumns": total_columns,
                "Count>0.05": exceed_count,
                "Percentage>0.05": percentage,
            })

# Convert results into a DataFrame
results_df = pd.DataFrame(results)

# Display the results
print(results_df)

## create tables from df
# Function to get row colors based on Percentage
def get_row_color(percentage):
    if percentage == 0:
        return "darkviolet"
    elif percentage < 50:
        return "lavender"
    elif percentage >= 50 and percentage < 100:
        return "lightyellow"
    elif percentage == 100:
        return "gold"
    return "white"

# Function to create a table in a specific subplot
def create_styled_table_in_subplot(ax, df, title="Table"):
    ax.axis("off")
    ax.set_title(title, fontsize=16, pad=26)

    # Add a table
    table = Table(ax, bbox=[0, 0, 1, 1])
    nrows, ncols = df.shape
```

```
        # Column headers
        col_labels = df.columns
        for col_idx, label in enumerate(col_labels):
            table.add_cell(-1, col_idx, text=label, width=1, height=0.2, facecolor=
            "lightgray", loc="center")

        # Row cells
        prev_subarea = None
        for row_idx, row in df.iterrows():
            current_subarea = row["Area"]
            edgecolor = "black"
            if prev_subarea != current_subarea:
                edgecolor = "black"

            prev_subarea = current_subarea

            for col_idx, value in enumerate(row):
                # Get cell color
                if col_labels[col_idx] == "Percentage>0.05":
                    cell_color = get_row_color(row["Percentage>0.05"])
                else:
                    cell_color = "white"

                table.add_cell(
                    row_idx,
                    col_idx,
                    text=str(value),
                    width=1,
                    height=0.2,
                    facecolor=cell_color,
                    loc="center",
                    edgecolor=edgecolor,
                )

        # Add the table to the subplot
        ax.add_table(table)

    # Prepare the DataFrame subsets and clean up slope names
    results_df_cleaned = results_df.drop(columns=["Slope"])  # Remove the Slope column
    unique_slopes = results_df["Slope"].unique()

    # Create subplots for the tables
    fig, axes = plt.subplots(1, len(unique_slopes), figsize=(24, 10))  # Increased size
    for higher resolution
    fig.tight_layout(pad=5)

    # Create a table for each slope type
    for ax, slope in zip(axes, unique_slopes):
        slope_df = results_df[results_df["Slope"] == slope].drop(columns=["Slope"])
        clean_title = f"Slope: {slope.replace('_slope', '')}"  # Clean slope name
        create_styled_table_in_subplot(ax, slope_df, title=clean_title)

    plt.show()

    ## Summary of table in percentages: Calculate if sub areas in general are more equal
    or more random, for each sub area seperatly if it's more equal or more random, and
    what slope is most equal or most random
    # Function to create the summary DataFrame with percentages
    def create_summary_percentage_df(results_df):
        # Initialize the empty DataFrame to store results
        summary_data = []

        # Total number of rows
        total_rows = len(results_df)

        # SubArea specific rows (as percentage)
        subareas = results_df["Area"].unique()
        for subarea in subareas:
            subarea_df = results_df[results_df["Area"] == subarea]
            subarea_total = len(subarea_df)  # Number of rows in the specific SubArea
            summary_data.append([f"{subarea}",
```

```
                                (subarea_df["Percentage>0.05"] == 0).sum() /
                                subarea_total * 100,
                                ((subarea_df["Percentage>0.05"] > 0) & (subarea_df[
                                "Percentage>0.05"] < 50)).sum() / subarea_total * 100,
                                ((subarea_df["Percentage>0.05"] >= 50) & (subarea_df[
                                "Percentage>0.05"] < 100)).sum() / subarea_total * 100,
                                (subarea_df["Percentage>0.05"] == 100).sum() /
                                subarea_total * 100])

    # Create DataFrame from the summary data
    summary_df = pd.DataFrame(summary_data, columns=["Area",
                                                     "Highly heterogeneous [%]",
                                                     "Heterogeneous [%]",
                                                     "Homogeneous [%]",
                                                     "Highly homogeneous [%]"])

    area_mapping = {
        "Area1": "Southern Taymyr",
        "Area2": "Northern Olenek",
        "Area3": "Chokurdakh",
        "Area4": "Iultinsky (Chukotka)",
        "Area6": "Southern Verkhoyansk Range"
    }

    # Replace area codes with names
    summary_df["Area"] = summary_df["Area"].replace(area_mapping)

    numeric_columns = ["Highly heterogeneous [%]", "Heterogeneous [%]", "Homogeneous
    [%]", "Highly homogeneous [%]"]
    summary_df[numeric_columns] = summary_df[numeric_columns].round(0).astype(int)


    return summary_df

# Create the summary DataFrame with percentages
summary_percentage_df = create_summary_percentage_df(results_df)

# Display the summary DataFrame with percentages
print(summary_percentage_df)

"""## Pie Charts of Homogeneity vers Heterogeneity"""

import matplotlib.pyplot as plt
import os
#from google.colab import drive
#drive.mount('/content/drive')

# Single pie charts for overview map
def save_single_pie_chart(summary_df, area_name, save_path):
    # Define the colors for each category
    colors = {
        "Highly heterogeneous [%]": "darkviolet",
        "Heterogeneous [%]": "lavender",
        "Homogeneous [%]": "lightyellow",
        "Highly homogeneous [%]": "gold"
    }

    # Filter data for the specified area
    row = summary_df[summary_df['Area'] == area_name].iloc[0]

    # Data for the pie chart
    labels = ["Highly heterogeneous [%]", "Heterogeneous [%]", "Homogeneous [%]",
    "Highly homogeneous [%]"]
    sizes = [row[label] for label in labels]

    # Filter out categories with zero values
    filtered_labels = [label for label, size in zip(labels, sizes) if size > 0]
    filtered_sizes = [size for size in sizes if size > 0]
    filtered_colors = [colors[label] for label in filtered_labels]

    # Create a figure and axis
    fig, ax = plt.subplots(figsize=(4, 4))
```

```
        # Create the pie chart
        wedges, texts, autotexts = ax.pie(filtered_sizes,  colors=filtered_colors,
               autopct='%1.0f%%', startangle=140, wedgeprops={'edgecolor': 'black'})

        ax.set_title(f"{area_name}", fontsize=18, weight="bold", y=0.95)

        for autotext in autotexts:
            autotext.set_fontsize(18)

        # Adjust layout for better spacing
        plt.tight_layout()

        # Save the plot to a file
        fig.savefig(save_path)
        print(f"Saved plot as {save_path}")

        # Show the plot
        plt.show()

titles = ["Southern Taymyr", "Northern Olenek", "Chokurdakh", "Iultinsky (Chukotka)",
"Southern Verkhoyansk Range"] #["Area 1", "Area 2", "Area 3", "Area 4", "Area 6"]

save_single_pie_chart(summary_percentage_df, "Southern Verkhoyansk Range",
'/content/drive/My Drive/Colab
Notebooks/Data/Plots/SouthernTaymyrSimilarityClassesPie.svg')

"""# Calculation of the impact of the 5 m buffer area

Calculation of buffer area
"""

import geopandas as gpd
from shapely.geometry import shape
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

"""This UTM zones correspond to the study areas

*   Area 1 - UTM Zone 47N (EPSG:32647)
*   Area 2 - UTM Zone 50N (EPSG:32650)
*   Area 3 - UTM Zone 54N (EPSG:32654)
*   Area 4 - UTM Zone 1N (EPSG:32601)
*   Area 6 - UTM Zone 51N (EPSG:32651)

"""

def gee_featurecollection_to_gdf(feature_collection, UTMz):

    # Convert FeatureCollection to a list
    feature_list = feature_collection.toList(feature_collection.size()).getInfo()

    gdf = gpd.GeoDataFrame(
        [
            {'id': feature['id'], 'geometry': shape(feature['geometry'])}
            for feature in feature_list
        ],
        geometry='geometry',
        crs="EPSG:4326"  # WGS 84 (Latitude/Longitude)
    )

    # Reproject to UTM zone for accurate area calculation
    gdf = gdf.to_crs(epsg=UTMz)

    # Compute original area in square kilometers
    gdf['area_m2'] = gdf['geometry'].area

    # Compute inward 5m buffer (negative buffer shrinks the polygon)
    gdf['buffer_5m'] = gdf['geometry'].buffer(-5)

    # Compute area of the inward 5m buffer (handle invalid geometries)
```

```
        gdf['area-5m_m2'] = gdf['buffer_5m'].apply(
            lambda geom: geom.area if geom.is_valid and not geom.is_empty else 0
        )
        gdf['area_5m_buffer_m2'] = gdf['area_m2'] - gdf['area-5m_m2']

     # Compute the percentage of the total area that was removed by buffering
        gdf['buffer_percentage'] = gdf.apply(
            lambda row: (row['area_5m_buffer_m2'] / row['area_m2']) * 100 if row['area_m2'] > 0 else 0,
            axis=1
        )

        # Add the updated geometry of the polygon after extracting the -5m buffer
        gdf['geometry2'] = gdf['buffer_5m']

        # Convert back to WGS 84 for geographic consistency
        gdf = gdf.to_crs(epsg=4326)

        return gdf.drop(columns=['buffer_5m'])

    rtsT1_gdf = gee_featurecollection_to_gdf(rtsT1, 32647)

    rtsT2_gdf = gee_featurecollection_to_gdf(rtsT2, 32650)

    rtsT3_gdf = gee_featurecollection_to_gdf(rtsT3, 32654)

    rtsT4_gdf = gee_featurecollection_to_gdf(rtsT4, 32601)

    rtsT6_gdf = gee_featurecollection_to_gdf(rtsT6, 32651)

    """Plotting Buffer Area with Histograms"""

    # Create a new column for each GeoDataFrame indicating the source
    rtsT1_gdf['area'] = 'T1'
    rtsT2_gdf['area'] = 'T2'
    rtsT3_gdf['area'] = 'T3'
    rtsT4_gdf['area'] = 'T4'
    rtsT6_gdf['area'] = 'T6'

    # Concatenate the buffer_percentage columns with the source column
    combined_df = pd.concat([
        rtsT1_gdf[['buffer_percentage', 'area']],
        rtsT2_gdf[['buffer_percentage', 'area']],
        rtsT3_gdf[['buffer_percentage', 'area']],
        rtsT4_gdf[['buffer_percentage', 'area']],
        rtsT6_gdf[['buffer_percentage', 'area']]
    ])

    # Dictionary to map abbreviations to full names
    area_mapping = {
        'T1': "Southern Taymyr",
        'T2': "Northern Olenek",
        'T3': "Chokurdakh",
        'T4': "Iultinsky (Chukotka)",
        'T6': "S. Verkhoyansk Range"
    }

    # Set up the FacetGrid with separate histograms for each source
    g = sns.FacetGrid(combined_df, col="area", col_wrap=5, height=4, aspect=1)

    # Plot the histograms in the individual plots and add a grey vertical line at x=20
    def plot_with_line(*args, **kwargs):
        # Plot the histogram without color argument
        sns.histplot(*args, **kwargs, kde=True)
        # Add the vertical line at x=20
        plt.axvline(x=20, color='grey', linestyle='--', linewidth=1)

    g.map(plot_with_line, 'buffer_percentage')

    # Manually update titles using the area_mapping dictionary
    for ax in g.axes.flat:
        # Get the current title (which is the 'area' value, e.g., 'T1')
```

```
        current_title = ax.get_title().split(' = ')[-1]  # Extract the area abbreviation
        # Map it to the full name and set the new title
        ax.set_title(area_mapping.get(current_title, current_title))

    # Set axis labels
    g.set_axis_labels('Buffer Percentage', 'Frequency')

    plt.tight_layout()

    # Show the plot
    plt.show()

    """Only area 4 seams to have large parts of the rtss beeing buffer zone.

    How much percent of RTSs per study area show an impact of the buffer area for more than 20% of their feature area?
    """

    # prompt: calculate the percentage (round (0)) for how much of the gdf the column gdf['buffer_percentage'] is higher than 20

    percentage_higher_than_20 = (rtsT1_gdf[rtsT1_gdf['buffer_percentage'] > 20].shape[0] / rtsT1_gdf.shape[0]) * 100
    rounded_percentage = round(percentage_higher_than_20, 0)
    print(f"{rounded_percentage}% of the gdf has a 'buffer_percentage' higher than 20.")

    """Percentage of RTSs of which their area is more than 20% coverdy by the 5 m buffer: T1 = 11%, T2 = 15%, T3 = 17% , T4 = 49%, T6 = 0%

    ## Apply new geometry (5m inward buffer) to RTSs of sub areas and repeat the Dunns test
    """

    import geopandas as gpd

    # Converting dataframes to GeoDataFrame
    gdf_T1_norm = gpd.GeoDataFrame(gdf_T1_norm, geometry='geometry')
    gdf_T2_norm = gpd.GeoDataFrame(gdf_T2_norm, geometry='geometry')
    gdf_T3_norm = gpd.GeoDataFrame(gdf_T3_norm, geometry='geometry')
    gdf_T4_norm = gpd.GeoDataFrame(gdf_T4_norm, geometry='geometry')
    gdf_T6_norm = gpd.GeoDataFrame(gdf_T6_norm, geometry='geometry')

    rtsT1_gdf = gpd.GeoDataFrame(rtsT1_gdf, geometry='geometry2')
    rtsT2_gdf = gpd.GeoDataFrame(rtsT2_gdf, geometry='geometry2')
    rtsT3_gdf = gpd.GeoDataFrame(rtsT3_gdf, geometry='geometry2')
    rtsT4_gdf = gpd.GeoDataFrame(rtsT4_gdf, geometry='geometry2')
    rtsT6_gdf = gpd.GeoDataFrame(rtsT6_gdf, geometry='geometry2')

    # Set the CRS to WGS 84 (EPSG:4326)
    gdf_T1_norm = gdf_T1_norm.set_crs(epsg=4326)
    gdf_T2_norm = gdf_T2_norm.set_crs(epsg=4326)
    gdf_T3_norm = gdf_T3_norm.set_crs(epsg=4326)
    gdf_T4_norm = gdf_T4_norm.set_crs(epsg=4326)
    gdf_T6_norm = gdf_T6_norm.set_crs(epsg=4326)

    rtsT1_gdf = rtsT1_gdf.to_crs(epsg=4326)
    rtsT2_gdf = rtsT2_gdf.to_crs(epsg=4326)
    rtsT3_gdf = rtsT3_gdf.to_crs(epsg=4326)
    rtsT4_gdf = rtsT4_gdf.to_crs(epsg=4326)
    rtsT6_gdf = rtsT6_gdf.to_crs(epsg=4326)

    # Create a mask to check if each point in gdf_T1_norm_2 is within any polygon in gdf['geometry2']
    mask = gdf_T1_norm['geometry'].apply(lambda point: rtsT1_gdf['geometry2'].apply(lambda poly: poly.contains(point)).any())
    gdf_T1_norm_2 = gdf_T1_norm[mask]

    print(len(gdf_T1_norm_2))
    print("Original number of rows:", len(gdf_T1_norm))

    mask = gdf_T2_norm['geometry'].apply(lambda point: rtsT2_gdf['geometry2'].apply(lambda poly: poly.contains(point)).any())
```

```
gdf_T2_norm_2 = gdf_T2_norm[mask]

print("New number of rows:",len(gdf_T2_norm_2))
print("Original number of rows:", len(gdf_T2_norm))

mask = gdf_T3_norm['geometry'].apply(lambda point: rtsT3_gdf['geometry2'].apply(lambda poly: poly.contains(point)).any())
gdf_T3_norm_2 = gdf_T3_norm[mask]

print("New number of rows:",len(gdf_T3_norm_2))
print("Original number of rows:", len(gdf_T3_norm))

mask = gdf_T4_norm['geometry'].apply(lambda point: rtsT4_gdf['geometry2'].apply(lambda poly: poly.contains(point)).any())
gdf_T4_norm_2 = gdf_T4_norm[mask]

print("New number of rows:",len(gdf_T4_norm_2))
print("Original number of rows:", len(gdf_T4_norm))

mask = gdf_T6_norm['geometry'].apply(lambda point: rtsT6_gdf['geometry2'].apply(lambda poly: poly.contains(point)).any())
gdf_T6_norm_2 = gdf_T6_norm[mask]

print("New number of rows:",len(gdf_T6_norm_2))
print("Original number of rows:", len(gdf_T6_norm))

subareas = {
    "SubAreas1": SubAreas1,
    "SubAreas2": SubAreas2,
    "SubAreas3": SubAreas3,
    "SubAreas4": SubAreas4,
    "SubAreas6": SubAreas6
}

gdfs = {
    "gdf_T1_norm": gdf_T1_norm_2,
    "gdf_T2_norm": gdf_T2_norm_2,
    "gdf_T3_norm": gdf_T3_norm_2,
    "gdf_T4_norm": gdf_T4_norm_2,
    "gdf_T6_norm": gdf_T6_norm_2
}

all_gdf_SubAreas = {}

# Loop through each subarea-feature-collection pair
for subarea_name, subarea_fc in subareas.items():
    # Get the corresponding GeoDataFrame for the subarea
    gdf_name = f"gdf_T{subarea_name[-1]}_norm"
    gdf = gdfs[gdf_name]

    gdf_dict = {}

    # Iterate through the features (squares) in the current subarea
    for i, square_feature in enumerate(subarea_fc.getInfo()['features']):
        # Extract the geometry of the current square
        square_geometry = shape(square_feature['geometry'])

        # Filter the points that fall within the current square
        filtered_points = gdf[gdf.geometry.apply(lambda geom: geom.within(
        square_geometry))]

        # Check if filtered_points is empty
        if filtered_points.empty:
            print(f"Warning: No points found in {subarea_name} square {i + 1}.
            Skipping this square.")
            continue  # Skip to the next square

        gdf_dict[f"{gdf_name}_{i + 1}"] = filtered_points

    all_gdf_SubAreas[subarea_name] = gdf_dict

    # Print keys of the current dictionary for verification
```

```
        print(f"Created GeoDataFrames for {subarea_name}:", list(gdf_dict.keys()))

from scipy.stats import shapiro
from scipy.stats import anderson
from scipy.stats import kruskal
!pip install scikit-posthocs
# during analysis the version "0.11.2-py3-none-any.whl.metadata (5.8 kB)" was used.
#That version is no longer availabe and version 0.11.3 can produce slightly different results.
import scikit_posthocs as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap, BoundaryNorm, Normalize

# Dunn
columns_to_test = ["TCB_slope", "TCG_slope", "TCW_slope"]

p_values_Dunn = {}

# Loop through each SubArea in all_gdf_SubAreas
for subarea_key, subarea_df_dict in all_gdf_SubAreas.items():
    print(f"\nPerforming Dunn's Test for {subarea_key}")

    p_values_Dunn[subarea_key] = {}

    # Loop through each column and perform Dunn's test
    for column in columns_to_test:
        print(f"  Testing column: {column}")

        # Create a list of the values for the column from each DataFrame in the current SubArea
        data = [df[column].dropna() for df_name, df in subarea_df_dict.items()]

        # Get dynamic sample labels based on available sub-areas
        sample_labels = [df_name.split('_')[-1] for df_name in subarea_df_dict.keys()]

        # Perform Dunn's test (pairwise comparisons)
        p_values = sp.posthoc_dunn(data, p_adjust="bonferroni")  # Using Bonferroni correction

        # Store the p-values as a DataFrame for the current column and SubArea
        p_values_Dunn[subarea_key][column] = pd.DataFrame(
            p_values.values,
            columns=sample_labels,
            index=sample_labels
        )

# To check the results for each SubArea and column, printing the p_values_Dunn dictionary
p_values_Dunn

# Custom colormap
colors = [
    (0.5, 0.1, 0.5),
    (1, 0.8, 0.4),      # Yellow starts at 0.05
    (1, 0.8, 0.4)
]
positions = [0.0000, 0.0500, 1]
cmap_name = "custom_gradient_cmap"
smooth_cmap = LinearSegmentedColormap.from_list(cmap_name, list(zip(positions, colors)))

# Normalize to align colors with specific ranges
norm = Normalize(vmin=0, vmax=1)

# Create a subplot grid with 1 row and number of columns based on the number of SubAreas
fig, axes = plt.subplots(5, 3, figsize=(20,20))

axes_flat = axes.flatten()
```

```
1489
1490  # Loop through the p_values_Dunn dictionary and plot each matrix
1491  k = 0
1492  for i, (subarea_key, subarea_p_values) in enumerate(p_values_Dunn.items()):
1493      for j, (column, matrix) in enumerate(subarea_p_values.items()):
1494          # Get the current subplot axis
1495          ax = axes_flat[k]
1496          k += 1
1497
1498          # Plot the heatmap for the current subarea and column
1499          sample_labels = list(matrix.columns)
1500
1501          # Plot heatmap for each SubArea and column
1502          sns.heatmap(
1503              matrix,
1504              annot=True,
1505              fmt=".4f",
1506              cmap=smooth_cmap,
1507              norm=norm,
1508              cbar_kws={'label': 'p-value'},
1509              ax=ax
1510          )
1511
1512          ax.set_title(f'Dunn-Bonferroni-Test P-Values: {subarea_key} - {column}')
1513          ax.set_xlabel('Sub Areas')
1514          ax.set_ylabel('Sub Areas')
1515
1516          # Get the current x-tick locations
1517          xticks = ax.get_xticks()
1518
1519          # Set x-tick labels only for the available tick locations
1520          ax.set_xticks(xticks)
1521          ax.set_xticklabels(sample_labels[:len(xticks)], rotation=0, ha='right')
1522
1523          # Similarly, for y-axis:
1524          yticks = ax.get_yticks()
1525          ax.set_yticks(yticks)
1526          ax.set_yticklabels(sample_labels[:len(yticks)], rotation=0)
1527  plt.tight_layout()
1528  #plt.savefig(f'/content/drive/My Drive/Colab
      Notebooks/Data/Dunn-Bonnferroni-Test_Sub-Areas.svg', format='svg')
1529  plt.show()
1530
1531  """## Analysis Dunn values -5 m buffer (homogeneity/heterogeneity)"""
1532
1533  import pandas as pd
1534  import matplotlib.pyplot as plt
1535  import matplotlib.colors as mcolors
1536  from matplotlib.table import Table
1537  import matplotlib.patches as patches
1538
1539  # df that contains information on sub areas exceeding 0.05 (-> are similar)
1540  data = p_values_Dunn
1541
1542  results = []
1543
1544  # Iterate through the subareas and slope types
1545  for subarea_key, slopes in data.items():
1546      for slope_key, matrix in slopes.items():
1547          # Compute row counts exceeding 0.05
1548          for row_index, row_values in matrix.iterrows():
1549              total_columns = len(row_values) -1
1550              exceed_count = (row_values > 0.05).sum() -1
1551              percentage = (exceed_count / total_columns) * 100
1552              modified_subarea_key = subarea_key[3:]  # Remove first three character
1553              modified_subarea_key = modified_subarea_key[:4] + modified_subarea_key[5:]
                   # Remove 8. character
1554              results.append({
1555                  "Area": modified_subarea_key,
1556                  "Slope": slope_key,
1557                  "SubAreaIndex": row_index,
1558                  "TotalColumns": total_columns,
```

```
                "Count>0.05": exceed_count,
                "Percentage>0.05": percentage,
            })

results_df = pd.DataFrame(results)

# Display the results
print(results_df)

## create tables from df
# Function to get row colors based on Percentage
def get_row_color(percentage):
    if percentage == 0:
        return "darkviolet"
    elif percentage < 50:
        return "lavender"
    elif percentage >= 50 and percentage < 100:
        return "lightyellow"
    elif percentage == 100:
        return "gold"
    return "white"

# Function to create a table in a specific subplot
def create_styled_table_in_subplot(ax, df, title="Table"):
    ax.axis("off")
    ax.set_title(title, fontsize=16, pad=26)

    table = Table(ax, bbox=[0, 0, 1, 1])
    nrows, ncols = df.shape

    # Column headers
    col_labels = df.columns
    for col_idx, label in enumerate(col_labels):
        table.add_cell(-1, col_idx, text=label, width=1, height=0.2, facecolor=
        "lightgray", loc="center")

    # Row cells
    prev_subarea = None
    for row_idx, row in df.iterrows():
        current_subarea = row["Area"]
        edgecolor = "black"
        if prev_subarea != current_subarea:
            edgecolor = "black"

        prev_subarea = current_subarea

        for col_idx, value in enumerate(row):
            # Get cell color
            if col_labels[col_idx] == "Percentage>0.05":
                cell_color = get_row_color(row["Percentage>0.05"])
            else:
                cell_color = "white"

            table.add_cell(
                row_idx,
                col_idx,
                text=str(value),
                width=1,
                height=0.2,
                facecolor=cell_color,
                loc="center",
                edgecolor=edgecolor,
            )

    ax.add_table(table)

# Prepare the DataFrame subsets and clean up slope names
results_df_cleaned = results_df.drop(columns=["Slope"])  # Remove the Slope column
unique_slopes = results_df["Slope"].unique()

# Create subplots for the tables
fig, axes = plt.subplots(1, len(unique_slopes), figsize=(24, 10))  # Increased size
```

```
    for higher resolution
1630 fig.tight_layout(pad=5)
1631
1632 # Create a table for each slope type
1633 for ax, slope in zip(axes, unique_slopes):
1634     slope_df = results_df[results_df["Slope"] == slope].drop(columns=["Slope"])
1635     clean_title = f"Slope: {slope.replace('_slope', '')}"  # Clean slope name
1636     create_styled_table_in_subplot(ax, slope_df, title=clean_title)
1637
1638 plt.show()
1639
1640 ## Summary of table in percentages: Calculate if sub areas in general are more equal
    or more random, for each sub area seperatly if it's more equal or more random, and
    what slope is most equal or most random
1641 # Function to create the summary DataFrame with percentages
1642 def create_summary_percentage_df(results_df):
1643     summary_data = []
1644
1645     total_rows = len(results_df)
1646
1647     # SubArea specific rows (as percentage)
1648     subareas = results_df["Area"].unique()
1649     for subarea in subareas:
1650         subarea_df = results_df[results_df["Area"] == subarea]
1651         subarea_total = len(subarea_df)
1652         summary_data.append([f"{subarea}",
1653                              (subarea_df["Percentage>0.05"] == 0).sum() /
                             subarea_total * 100,
1654                              ((subarea_df["Percentage>0.05"] > 0) & (subarea_df[
                             "Percentage>0.05"] < 50)).sum() / subarea_total * 100,
1655                              ((subarea_df["Percentage>0.05"] >= 50) & (subarea_df[
                             "Percentage>0.05"] < 100)).sum() / subarea_total * 100,
1656                              (subarea_df["Percentage>0.05"] == 100).sum() /
                             subarea_total * 100])
1657
1658     summary_df = pd.DataFrame(summary_data, columns=["Area",
1659                                                      "Highly heterogeneous [%]",
1660                                                      "Heterogeneous [%]",
1661                                                      "Homogeneous [%]",
1662                                                      "Highly homogeneous [%]"])
1663     area_mapping = {
1664         "Area1": "Southern Taymyr",
1665         "Area2": "Northern Olenek",
1666         "Area3": "Chokurdakh",
1667         "Area4": "Iultinsky (Chukotka)",
1668         "Area6": "Southern Verkhoyansk Range"
1669     }
1670
1671     # Replace area codes with names
1672     summary_df["Area"] = summary_df["Area"].replace(area_mapping)
1673
1674     numeric_columns = ["Highly heterogeneous [%]", "Heterogeneous [%]", "Homogeneous
    [%]", "Highly homogeneous [%]"]
1675     summary_df[numeric_columns] = summary_df[numeric_columns].round(0).astype(int)
1676
1677
1678     return summary_df
1679
1680 # Create the summary DataFrame with percentages
1681 summary_percentage_df = create_summary_percentage_df(results_df)
1682
1683 # Display the summary DataFrame with percentages
1684 print(summary_percentage_df)
1685
1686 # pie charts for overview map
1687 def save_single_pie_chart(summary_df, area_name, save_path):
1688     # Define the colors for each category
1689     colors = {
1690         "Highly heterogeneous [%]": "darkviolet",
1691         "Heterogeneous [%]": "lavender",
1692         "Homogeneous [%]": "lightyellow",
1693         "Highly homogeneous [%]": "gold"
```

```
    }

    # Filter data for the specified area
    row = summary_df[summary_df['Area'] == area_name].iloc[0]

    # Data for the pie chart
    labels = ["Highly heterogeneous [%]", "Heterogeneous [%]", "Homogeneous [%]",
    "Highly homogeneous [%]"]
    sizes = [row[label] for label in labels]

    # Filter out categories with zero values
    filtered_labels = [label for label, size in zip(labels, sizes) if size > 0]
    filtered_sizes = [size for size in sizes if size > 0]
    filtered_colors = [colors[label] for label in filtered_labels]

    # Create a figure and axis
    fig, ax = plt.subplots(figsize=(4, 4))

    # Create the pie chart
    wedges, texts, autotexts = ax.pie(filtered_sizes,  colors=filtered_colors,
           autopct='%1.0f%%', startangle=140, wedgeprops={'edgecolor': 'black'},
           pctdistance=0.83)

    ax.set_title(f"{area_name}", fontsize=18, weight="bold", y=0.95)

    for autotext in autotexts:
        autotext.set_fontsize(18)

    plt.tight_layout()

    # Save the plot to a file
    fig.savefig(save_path)
    print(f"Saved plot as {save_path}")

    # Show the plot
    plt.show()

titles = ["Southern Taymyr", "Iultinsky (Chukotka)"] #["Area 1", "Area 2", "Area 3",
"Area 4", "Area 6"]

save_single_pie_chart(summary_percentage_df, "Iultinsky (Chukotka)",
'/content/drive/My Drive/Colab
Notebooks/Data/Plots/SouthernTaymyrSimilarityClassesPie.svg')

"""# Terrain position (TP) detection"""

import pandas as pd
import geopandas as gpd

"""Converting the lakes data set to shp file and storring it as gee asset"""

#Lakes = gpd.read_parquet('/content/drive/My Drive/Colab
Notebooks/Data/filtered_full_set_v2.parquet')
#Lakes_gdf = gpd.GeoDataFrame(Lakes, geometry="geometry", crs="EPSG:4326")
#Lakes_gdf.to_file("/content/drive/My Drive/Colab Notebooks/Data/Lakes.shp",
driver="ESRI Shapefile")

"""## Shore line"""

# Install required libraries
!pip install geemap geopandas

import geemap
import geopandas as gpd
import pandas as pd

# Load the FeatureCollections

MainlandPolygon = ee.FeatureCollection(
'projects/sat-io/open-datasets/shoreline/mainlands')

def check_shore_overlap(feature):
```

```
        # Find polygons that intersect with the feature
        intersects = MainlandPolygon.filterBounds(feature.geometry())

        # Get the first intersecting polygon (or null if none found)
        first_intersecting_polygon = intersects.first()

        # Check if the feature is fully contained within the mainland polygon
        fully_contained = ee.Algorithms.If(
            first_intersecting_polygon,  # Condition: if intersecting polygon exists
            first_intersecting_polygon.geometry().contains(feature.geometry()),  # If true: perform 'contains'
            False  # If false: assume 'not fully overlapping' (set to False)
        )

        # Assigns 0 (fully overlaps) or 1 (not fully overlapping) to TP property
        updated_feature = feature.set("TP", ee.Algorithms.If(fully_contained, 0, 1))

        # Select only relevant properties (mimics Pandas .loc[:, columns_to_keep])
        columns_to_keep = ["geometry", "TP", "id", "fid"]
        selected_properties = updated_feature.select(columns_to_keep)

        return selected_properties

    # Apply the function shore overlap to Area 1
    TP_rtsT1 = rtsT1.map(check_shore_overlap)

    # Convert to GeoJSON
    geojson_TP_rtsT1 = geemap.ee_to_geojson(TP_rtsT1)
    # Convert to GeoDataFrame
    gdf_TP_rtsT1 = gpd.GeoDataFrame.from_features(geojson_TP_rtsT1)

    print("Number of rows where TP == 1:", len(gdf_TP_rtsT1[gdf_TP_rtsT1['TP'] == 1]))

    # Apply the function shore overlap to Area 2
    TP_rtsT2 = rtsT2.map(check_shore_overlap)

    # Convert to GeoJSON
    geojson_TP_rtsT2 = geemap.ee_to_geojson(TP_rtsT2)
    # Convert to GeoDataFrame
    gdf_TP_rtsT2 = gpd.GeoDataFrame.from_features(geojson_TP_rtsT2)

    print("Number of rows where TP == 1:", len(gdf_TP_rtsT2[gdf_TP_rtsT2['TP'] == 1]))

    # Apply the function shore overlap to Area 4
    TP_rtsT4 = rtsT4.map(check_shore_overlap)

    # Convert to GeoJSON
    geojson_TP_rtsT4 = geemap.ee_to_geojson(TP_rtsT4)
    # Convert to GeoDataFrame
    gdf_TP_rtsT4 = gpd.GeoDataFrame.from_features(geojson_TP_rtsT4)

    print("Number of rows where TP == 1:", len(gdf_TP_rtsT4[gdf_TP_rtsT4['TP'] == 1]))

    # Apply the function shore overlap to Area 6
    TP_rtsT6 = rtsT6.map(check_shore_overlap)

    # Convert to GeoJSON
    geojson_TP_rtsT6 = geemap.ee_to_geojson(TP_rtsT6)
    # Convert to GeoDataFrame
    gdf_TP_rtsT6 = gpd.GeoDataFrame.from_features(geojson_TP_rtsT6)

    print("Number of rows where TP == 1:", len(gdf_TP_rtsT6[gdf_TP_rtsT6['TP'] == 1]))

    # creating same gdf for Area 3
    def add_tp_column(feature):
      """Adds a TP property to the feature and sets its value to 0."""
      return feature.set('TP', 0)

    # Map the function to the FeatureCollection
    rtsT3_with_tp = rtsT3.map(add_tp_column)
```

```
# Select only the desired columns
rtsT3_selected = rtsT3_with_tp.select(['geometry', 'id', 'fid', 'TP'])
# Convert the FeatureCollection to a GeoDataFrame
geojson_TP_rtsT3 = geemap.ee_to_geojson(rtsT3_selected)
gdf_TP_rtsT3 = gpd.GeoDataFrame.from_features(geojson_TP_rtsT3)

# Display the first few rows to verify
print(gdf_TP_rtsT3.head())

"""Check results visualy"""

#prepaire data for visualization
gdf_TP_rtsT2.set_crs(epsg=4326, inplace=True)  # Set CRS directly on gdf_TP_rtsT2 with inplace=True
TP_rtsT2_fc = geemap.geopandas_to_ee(gdf_TP_rtsT2)
TP_rtsT2_fc_2 = TP_rtsT2_fc.filter(ee.Filter.eq('TP', 1))

polygon_style = {
    'color': 'red',
    'width': 2,
    'fillColor': '00000000'  # Transparent fill
}

Map = geemap.Map(center=[73.25, 116.5], zoom=5)

# Add the filtered polygons (only overlap = 1)
Map.addLayer(TP_rtsT2_fc_2.style(**polygon_style), {}, "Filtered Polygons (overlap = 1)")

# Display map
Map

"""## Lakes"""

Lakes = ee.FeatureCollection("projects/ee-moritzjulia7/assets/Lakes")

# Define FeatureCollections
rtsT_collections = {
    "rtsT1": rtsT1,
    "rtsT2": rtsT2,
    "rtsT3": rtsT3,
    "rtsT4": rtsT4,
    "rtsT6": rtsT6
}

gdf_TP_dict = {
    "rtsT1": gdf_TP_rtsT1,
    "rtsT2": gdf_TP_rtsT2,
    "rtsT3": gdf_TP_rtsT3,
    "rtsT4": gdf_TP_rtsT4,
    "rtsT6": gdf_TP_rtsT6
}

# Function to check intersection in GEE and update TP value
def update_tp_if_intersects(rts_fc, gdf_TP):
    def check_overlap(feature):
        """Check if the feature intersects with any polygon in Lakes."""
        intersects = Lakes.filterBounds(feature.geometry()).size().gt(0)
        return feature.set("TP", ee.Algorithms.If(intersects, 2, feature.get("TP")))

    # Apply intersection check to each polygon in the rtsT FeatureCollection
    updated_fc = rts_fc.map(check_overlap)

    # Convert updated FeatureCollection to a Pandas DataFrame
    updated_gdf = geemap.ee_to_geojson(updated_fc)
    updated_gdf = gpd.GeoDataFrame.from_features(updated_gdf)

    # Merge to keep original structure but update TP where needed
    gdf_TP.set_index("fid", inplace=True)
    updated_gdf.set_index("fid", inplace=True)

    # Update only the TP values
```

```
        gdf_TP.update(updated_gdf["TP"])

        # Reset index after updating
        gdf_TP.reset_index(inplace=True)

        return gdf_TP

    # Process all rtsT datasets
    for key, rts_fc in rtsT_collections.items():
        gdf_TP_dict[key] = update_tp_if_intersects(rts_fc, gdf_TP_dict[key])

    """Check results visualy"""

    #prepaire data for visualization
    gdf_TP_rtsT2.set_crs(epsg=4326, inplace=True)  # Set CRS directly on gdf_TP_rtsT2 with inplace=True
    TP_rtsT2_fc = geemap.geopandas_to_ee(gdf_TP_rtsT2)
    TP_rtsT2_fc_2 = TP_rtsT2_fc.filter(ee.Filter.eq('TP', 2))

    polygon_style = {
        'color': 'red',
        'width': 2,
        'fillColor': '00000000'  # Transparent fill
    }

    Map = geemap.Map(center=[73.25, 116.5], zoom=5)

    # Add the filtered polygons (only overlap = 2)
    Map.addLayer(TP_rtsT2_fc_2.style(**polygon_style), {}, "Filtered Polygons (overlap = 2)")

    # Display map
    Map

    """Downloading the data frames"""

    from google.colab import drive
    import os
    import shutil

    # Mount Google Drive
    #drive.mount('/content/drive')

    # Define the directory in Google Drive where files will be saved
    #save_dir = "/content/drive/My Drive/Colab Notebooks/Data/TP shp Verion 1"

    # Ensure the directory exists
    #os.makedirs(save_dir, exist_ok=True)

    # Loop through each GeoDataFrame and save as a Shapefile
    #for name, gdf in gdf_TP_dict.items():
    #    shp_dir = os.path.join(save_dir, name)  # Each shapefile needs its own folder
    #    os.makedirs(shp_dir, exist_ok=True)  # Create a folder for the shapefile components#

    #    file_path = os.path.join(shp_dir, name + ".shp")
    #    gdf.to_file(file_path, driver="ESRI Shapefile")

    #    print(f"Saved {name} as a Shapefile in {shp_dir}")

    """## Loading the manually enhanced data sets"""

    from google.colab import drive
    import geopandas as gpd
    import geemap

    drive.mount('/content/drive')

    rtsT1_v2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/TP shp version 2/rtsT1_v2.shp' )
    rtsT2_v2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/TP shp version 2/rtsT2_v2.shp')
```

```
rtsT3_v2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/TP shp version 2/rtsT3_v2.shp')
rtsT4_v2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/TP shp version 2/rtsT4_v2.shp')
rtsT6_v2 = gpd.read_file('/content/drive/My Drive/Colab Notebooks/Data/TP shp version 2/rtsT6_v2.shp')

"""## Analyse TP Data"""

import geopandas as gpd
import pandas as pd

## Merging the TP data frames to one large data frame
gdf_list = [
    (rtsT1_v2, 1),
    (rtsT2_v2, 2),
    (rtsT3_v2, 3),
    (rtsT4_v2, 4),
    (rtsT6_v2, 6),
]

# Create a list of GeoDataFrames with the added 'area' column
gdfs_with_area = [
    gdf[['fid', 'TP', 'Shpe', 'geometry']].assign(area=area)
    for gdf, area in gdf_list
]

# Merge them into a single GeoDataFrame
rts_TP_all = gpd.GeoDataFrame(pd.concat(gdfs_with_area, ignore_index=True))

# Check the result
print(rts_TP_all)

import matplotlib.pyplot as plt

def create_tp_pie_charts(rts_TP_all, save_path=None):

    tp_labels = {
        1: "Sea Shore",
        2: "Lake Shore",
        3: "River Shore",
        4: "Gully",
       # 5: "Others",
        6: "Ponds (+Gully)"
    }

    tp_colors = {
        1: '#00008B',  # Dark Blue
        2: '#0000FF',  # Blue
        3: '#ADD8E6',  # Light Blue
        4: '#FFA500',  # Orange
       # 5: '#FF0000',   # Red
        6: '#800000',  # Maroon
    }

    area_name_mapping = {
         1: "Southern Taymyr",
         2: "Northern Olenek",
         3: "Chokurdakh",
         4: "Iultinsky (Chukotka)",
         6: "Southern Verkhoyansk Range"
    }

    # Replace area names in the 'area' column using the mapping
    rts_TP_all['area'] = rts_TP_all['area'].replace(area_name_mapping)

    # Get the unique areas
    unique_areas = rts_TP_all['area'].unique()
    num_areas = len(unique_areas)

    # Create subplots: arrange them in a row
    fig, axes = plt.subplots(1, num_areas, figsize=(6 * num_areas, 6))
```

```
    # If only one area
    if num_areas == 1:
        axes = [axes]

    # Iterate through each area and create a pie chart
    for idx, area in enumerate(unique_areas):
        area_df = rts_TP_all[rts_TP_all['area'] == area]
        tp_counts = area_df['TP'].value_counts()

        # Get sizes and colors
        sizes = tp_counts.values
        colors = [tp_colors[tp] for tp in tp_counts.index]

        # Create the pie chart
        wedges, _, autotexts = axes[idx].pie(
            sizes, autopct='%1.0f%%', colors=colors,
            startangle=140, wedgeprops={'edgecolor': 'black'},
            pctdistance=1.2  # Adjust percentage placement
        )

        axes[idx].set_title(f"{area}", fontsize=24, y=1.05)

        for autotext in autotexts:
            autotext.set_fontsize(18)

    # Create a shared legend at the bottom
    legend_labels = [f"{tp_labels[tp]}" for tp in tp_labels]
    legend_colors = [tp_colors[tp] for tp in tp_labels]

    # Create legend patches
    legend_patches = [plt.Line2D([0], [0], marker='o', color='w',
                                 markerfacecolor=color, markersize=12) for color in
                                 legend_colors]

    # Add legend below all plots
    fig.legend(legend_patches, legend_labels, loc="lower center",
               fontsize=18, ncol=5, bbox_to_anchor=(0.5, -0.05))

    # Adjust layout for better spacing
    plt.tight_layout(rect=[0, 0.1, 1, 1])

    # Set a bold title for the entire figure
    fig.suptitle(
        "Distribution of TP Values Across Areas",
        fontsize=28, weight="bold", y=1.12
    )

    if save_path:
        plt.savefig(save_path, bbox_inches="tight")
        print(f"Saved plot as {save_path}")

    plt.show()

create_tp_pie_charts(rts_TP_all)

"""## Analyse morphology Data"""

import geopandas as gpd
import pandas as pd

import matplotlib.pyplot as plt

def create_morphology_pie_charts(rts_TP_all, save_path=None):

    morphology_labels = {
        0: "Thermocirque",
        1: "Thermoterrace",
        2: "Combination",
    }

    m_colors = {
```

```
        0: '#800000',
        1: '#5FAF00',
        2: '#BEAA3C'
    }

    area_name_mapping = {
         1: "Southern Taymyr",
         2: "Northern Olenek",
         3: "Chokurdakh",
         4: "Iultinsky (Chukotka)",
         6: "Southern Verkhoyansk Range"
    }

    # Replace area names in the 'area' column using the mapping
    rts_TP_all['area'] = rts_TP_all['area'].replace(area_name_mapping)

    # Get the unique areas
    unique_areas = rts_TP_all['area'].unique()
    num_areas = len(unique_areas)

    # Create subplots: arrange them in a row
    fig, axes = plt.subplots(1, num_areas, figsize=(6 * num_areas, 6))

    # If only one area
    if num_areas == 1:
        axes = [axes]

    # Iterate through each area and create a pie chart
    for idx, area in enumerate(unique_areas):
        area_df = rts_TP_all[rts_TP_all['area'] == area]
        m_counts = area_df['Shpe'].value_counts()

        # Get sizes and colors
        sizes = m_counts.values
        colors = [m_colors[m] for m in m_counts.index]

        # Create the pie chart
        wedges, _, autotexts = axes[idx].pie(
            sizes, autopct='%1.0f%%', colors=colors,
            startangle=140, wedgeprops={'edgecolor': 'black'},
            pctdistance=1.2
        )

        axes[idx].set_title(f"{area}", fontsize=24, y=1.05)

        for autotext in autotexts:
            autotext.set_fontsize(18)

    # Create a shared legend at the bottom
    legend_labels = [f"{morphology_labels[m]}" for m in morphology_labels]
    legend_colors = [m_colors[m] for m in morphology_labels]

    # Create legend patches
    legend_patches = [plt.Line2D([0], [0], marker='o', color='w',
                                 markerfacecolor=color, markersize=12) for color in
                                 legend_colors]

    # Add legend below all plots
    fig.legend(legend_patches, legend_labels, loc="lower center",
               fontsize=18, ncol=5, bbox_to_anchor=(0.5, -0.05))

    # Adjust layout for better spacing
    plt.tight_layout(rect=[0, 0.1, 1, 1])

    fig.suptitle(
        "Distribution of Morphology Types Across Study Areas",
        fontsize=28, weight="bold", y=1.12
    )

    if save_path:
        plt.savefig(save_path, bbox_inches="tight")
        print(f"Saved plot as {save_path}")
```

```
    plt.show()

create_morphology_pie_charts(rts_TP_all)

"""# Terrain Position, spectral slope similarity compairison (sub-hypothesis 3)

Singe gdf creation per RTS. Names of gdf include value of main area, slumps fid, TP value and Shape(=Morphology) value
"""

import geopandas as gpd
import pandas as pd
from collections import defaultdict

polygon_layers = {
    "rtsT1_v2": rtsT1_v2,
    "rtsT2_v2": rtsT2_v2,
    "rtsT3_v2": rtsT3_v2,
    "rtsT4_v2": rtsT4_v2,
    "rtsT6_v2": rtsT6_v2
}

point_layers = {
    "gdf_T1_norm": gdf_T1_norm_2, #_2 are the df that have the 5 m inward buffer already applyed
    "gdf_T2_norm": gdf_T2_norm_2,
    "gdf_T3_norm": gdf_T3_norm_2,
    "gdf_T4_norm": gdf_T4_norm_2,
    "gdf_T6_norm": gdf_T6_norm_2
}

gdf_singleRTSs_plus_Property = {}

# Iterate through each polygon layer
for poly_name, poly_gdf in polygon_layers.items():
    T_number = poly_name[4]  # Extract T number (e.g., '1' from 'rtsT1_v2')
    point_gdf_name = f"gdf_T{T_number}_norm"

    if point_gdf_name not in point_layers:
        print(f"Warning: No matching point layer found for {poly_name}. Skipping.")
        continue

    point_gdf = point_layers[point_gdf_name]

    gdf_dict = {}

    # Iterate over each polygon in the layer
    for idx, polygon in poly_gdf.iterrows():
        polygon_geometry = polygon.geometry
        TP_value = polygon["TP"]
        fid = polygon["fid"]
        shape_number = polygon["Shpe"]

        # Filter points within the polygon
        filtered_points = point_gdf[point_gdf.geometry.apply(lambda geom: geom.within(polygon_geometry))]


        # Check if filtered points exist
        if filtered_points.empty:
            print(f"Warning: No points found in {poly_name} for polygon {fid}. Skipping.")
            continue

        # Create the dictionary key name
        gdf_key = f"T{T_number}_fid{fid}_TP{TP_value}_Shape{shape_number}"

        # Store in dictionary
        gdf_dict[gdf_key] = filtered_points

    # Store the dictionary for this polygon layer
```

```
        gdf_singleRTSs_plus_Property[f"T{T_number}"] = gdf_dict

        print(f"Processed {poly_name}: {len(gdf_dict)} RTS dfs created.")

    #Overview of created geodataframes

    summary_data = defaultdict(lambda: {"Total_GDFs": 0})

    # Iterate over all created GDFs
    for T_key, sub_gdfs in gdf_singleRTSs_plus_Property.items():
        for gdf_name in sub_gdfs.keys():
            # Extract TP value and Shape number from the GDF name
            parts = gdf_name.split("_")
            TP_value = int(parts[2][2:])    # Extract TP value from 'TPX'
            Shape_value = int(parts[3][5:])  # Extract Shape number from 'ShapeX'

            # Increment counts
            summary_data[T_key][f"TP_{TP_value}"] = summary_data[T_key].get(f"TP_{TP_value}", 0) + 1
            summary_data[T_key][f"Shape_{Shape_value}"] = summary_data[T_key].get(f"Shape_{Shape_value}", 0) + 1
            summary_data[T_key]["Total_GDFs"] += 1

    # Convert to Pandas DataFrame
    summary_df = pd.DataFrame.from_dict(summary_data, orient="index").fillna(0)

    # Display the summary table
    print(summary_df)

    """**Areas:** "T1": "Southern Taymyr", "T2": "Northern Olenek", "T3": "Chokurdakh", "T4": "Iultinsky (Chukotka)", "T6": "Southern Verkhoyansk Range"

    **Terrain Position Names:** "TP_1": "Sea", "TP_2": "Lake", "TP_3": "River", "TP_4": "Gully", "TP_5": "Others",  "TP_6": "Ponds + Gully"

    **Morphology Names:** "Shape_0": "Thermocirque", "Shape_1": "Thermoterrace", "Shape_2": "Combination"

    ## Calcutaion of statistics for all slumps
    """

    from scipy.stats import shapiro
    !pip install scikit-posthocs
    # during analysis the version "0.11.2-py3-none-any.whl.metadata (5.8 kB)" was used.
    #That version is no longer availabe and version 0.11.3 can produce slightly different results.
    import scipy.stats as stats
    import scikit_posthocs as sp
    import pandas as pd

    # shapiro -> normal distribution

    # Define the numerical columns to check for normality
    columns = ["TCB_slope", "TCG_slope", "TCW_slope"]

    shapiro_results = {}

    # Iterate through all areas (T1, T2, etc.)
    for T_key, sub_gdfs in gdf_singleRTSs_plus_Property.items():
        shapiro_results[T_key] = {}

        # Iterate through all created GeoDataFrames
        for df_name, df in sub_gdfs.items():
            shapiro_results[T_key][df_name] = {}

            for col in columns:
                try:
                    # Perform Shapiro-Wilk test only if the column exists
                    if col in df.columns and len(df[col].dropna()) > 3:
                        stat, p_value = shapiro(df[col].dropna())  # Remove NaN values
                        shapiro_results[T_key][df_name][col] = {"statistic": stat, "p_value": p_value}
```

```
                else:
                    shapiro_results[T_key][df_name][col] = {"statistic": None,
                    "p_value": None, "error": "Insufficient data"}
            except Exception as e:
                # Handle errors (e.g., not enough data points)
                shapiro_results[T_key][df_name][col] = {"statistic": None, "p_value":
                None, "error": str(e)}

# Print the results in a readable format
for T_key, subarea_results in shapiro_results.items():
    print(f"\n📊 Shapiro-Wilk Test Results for {T_key}:")
    for df_name, results in subarea_results.items():
        print(f"  🗁 DataFrame: {df_name}")
        for col, result in results.items():
            if result["statistic"] is not None:
                print(f"    {col}: statistic={result['statistic']:.4f}, p-value={
                result['p_value']:.4f}")
            else:
                print(f"    {col}: ✕ Test could not be performed. Error: {result[
                'error']}")

"""Majority of RTSs are **not** normaly distributed.


"""

### Dunn's Test and Kruskal-Wallis tests

# Define numerical columns to compare
columns = ["TCB_slope", "TCG_slope", "TCW_slope"]

comparison_results = {}

# Group GDFs by T number (Area) and TP value
for T_key, sub_gdfs in gdf_singleRTSs_plus_Property.items():
    tp_groups = {}  # Dictionary to store groups by TP value

    # Organize GDFs by TP value
    for df_name, df in sub_gdfs.items():
        parts = df_name.split("_")
        TP_value = int(parts[2][2:])  # Extract TP value from 'TPX'

        if TP_value not in tp_groups:
            tp_groups[TP_value] = []

        tp_groups[TP_value].append(df)

    # Perform statistical comparisons within each TP group
    for TP_value, gdf_list in tp_groups.items():
        if len(gdf_list) > 1:  # Only compare if we have multiple GDFs
            comparison_results[f"T{T_key}_TP{TP_value}"] = {}

            for col in columns:
                # Combine data from all GDFs in this group
                combined_data = [gdf[col].dropna().values for gdf in gdf_list]

                # Kruskal-Wallis test (checks if there's any difference)
                H_stat, p_kw = stats.kruskal(*combined_data)

                if p_kw < 0.05:  # If significant, perform Dunn's test
                    dunn_results = sp.posthoc_dunn(combined_data, p_adjust=
                    "bonferroni")
                    comparison_results[f"T{T_key}_TP{TP_value}"][col] = {"Kruskal_p":
                    p_kw, "Dunn": dunn_results}
                else:
                    comparison_results[f"T{T_key}_TP{TP_value}"][col] = {"Kruskal_p":
                    p_kw, "Dunn": None}

# Print results
for group, results in comparison_results.items():
    print(f"\n📊 Statistical Comparisons for {group}:")
    for col, stats_dict in results.items():
```

```
            print(f"  ◈ {col}: Kruskal-Wallis p-value = {stats_dict['Kruskal_p']:.4f}")
            if stats_dict["Dunn"] is not None:
                print(f"     Dunn's test results:\n{stats_dict['Dunn']}")
            else:
                print("     No significant differences found.")

"""## Homogenitiy classification"""

import matplotlib.pyplot as plt
from matplotlib.table import Table

results = []

# Iterate over each group (T, TP)
for group, group_results in comparison_results.items():
    # Extract T and TP from the group string
    T_value, TP_value = group.split('_')[0][1:], group.split('_')[1][2:]

    # Process each column in the group's results
    for slope_key, stats_dict in group_results.items():
        # If Dunn's test is available
        if stats_dict['Dunn'] is not None:
            matrix = stats_dict['Dunn']

            # Iterate over the rows of the matrix
            for row_index, row_values in matrix.iterrows():
                total_columns = len(row_values) - 1  # Subtracting 1 to exclude the comparison with itself
                exceed_count = (row_values > 0.05).sum() - 1  # Count values exceeding 0.05
                percentage = (exceed_count / total_columns) * 100

                # Append the result to the results list
                results.append({
                    "Area": T_value,  # Extracted T value
                    "TP": TP_value,  # Extracted TP value
                    "Slope": slope_key,  # The slope key (e.g., 'TCB_slope')
                    "RTSIndex": row_index,  # The row index (single slump index)
                    "TotalColumns": total_columns,  # Total columns in the matrix (excluding the diagonal)
                    "Count>0.05": exceed_count,  # Count of values exceeding 0.05
                    "Percentage>0.05": percentage  # Percentage of values exceeding 0.05
                })

# Print the results (or you could save them to a DataFrame)
results_df = pd.DataFrame(results)
print(results_df)

## create tables from df, cecking if patterns are visible
# Function to get row colors based on Percentage
def get_row_color(percentage):
    if percentage == 0:
        return "darkviolet"
    elif percentage < 50:
        return "lavender"
    elif percentage >= 50 and percentage < 100:
        return "lightyellow"
    elif percentage == 100:
        return "gold"
    return "white"

# Function to create a table in a specific subplot
def create_styled_table_in_subplot(ax, df, title="Table"):
    ax.axis("off")
    ax.set_title(title, fontsize=16, pad=26)

    table = Table(ax, bbox=[0, 0, 1, 1])
    nrows, ncols = df.shape

    col_labels = df.columns
    for col_idx, label in enumerate(col_labels):
```

```
            table.add_cell(-1, col_idx, text=label, width=1, height=0.2, facecolor=
            "lightgray", loc="center")

        # Row cells
        prev_subarea = None
        for row_idx, row in df.iterrows():
            current_subarea = row["Area"]
            edgecolor = "black"
            if prev_subarea != current_subarea:
                edgecolor = "black"

            prev_subarea = current_subarea

            for col_idx, value in enumerate(row):
                # Get cell color
                if col_labels[col_idx] == "Percentage>0.05":
                    cell_color = get_row_color(row["Percentage>0.05"])
                else:
                    cell_color = "white"

                table.add_cell(
                    row_idx,
                    col_idx,
                    text=str(value),
                    width=1,
                    height=0.2,
                    facecolor=cell_color,
                    loc="center",
                    edgecolor=edgecolor,
                )

        # Add the table to the subplot
        ax.add_table(table)

    # Prepare the DataFrame subsets and clean up slope names
    results_df_cleaned = results_df.drop(columns=["Slope"])  # Remove the Slope column
    unique_slopes = results_df["Slope"].unique()

    # Create subplots for the tables
    fig, axes = plt.subplots(1, len(unique_slopes), figsize=(54, 40))  # Increased size
    for higher resolution
    fig.tight_layout(pad=5)

    # Create a table for each slope type
    for ax, slope in zip(axes, unique_slopes):
        slope_df = results_df[results_df["Slope"] == slope].drop(columns=["Slope"])
        clean_title = f"Slope: {slope.replace('_slope', '')}"  # Clean slope name
        create_styled_table_in_subplot(ax, slope_df, title=clean_title)

    plt.show()

    def create_tp_summary_percentage_df(results_df):
        summary_dfs = {}  # Dictionary to store summary DataFrames for each TP

        # Get unique TP values
        tp_values = results_df["TP"].unique()

        for tp in tp_values:
            # Filter for current TP
            tp_df = results_df[results_df["TP"] == tp]

            summary_data = []
            areas = tp_df["Area"].unique()

            for area in areas:
                area_df = tp_df[tp_df["Area"] == area]
                area_total = len(area_df)

                summary_data.append([
                    f"{area}",
                    (area_df["Percentage>0.05"] == 0).sum() / area_total * 100,  # Highly
                    homogeneous
```

```
                    ((area_df["Percentage>0.05"] > 0) & (area_df["Percentage>0.05"] < 50
                    )).sum() / area_total * 100,  # Homogeneous
                    ((area_df["Percentage>0.05"] >= 50) & (area_df["Percentage>0.05"] <
                    100)).sum() / area_total * 100,  # Heterogeneous
                    (area_df["Percentage>0.05"] == 100).sum() / area_total * 100  #
                    Highly heterogeneous
                ])

            summary_df = pd.DataFrame(summary_data, columns=[
                "Area",
                "Highly heterogeneous [%]",
                "Heterogeneous [%]",
                "Homogeneous [%]",
                "Highly homogeneous [%]"
            ])

            area_mapping = {
                "T1": "Southern Taymyr",
                "T2": "Northern Olenek",
                "T3": "Chokurdakh",
                "T4": "Iultinsky (Chukotka)",
                "T6": "Southern Verkhoyansk Range"
            }

            summary_df["Area"] = summary_df["Area"].replace(area_mapping)

            # Round numeric columns
            numeric_cols = ["Highly homogeneous [%]", "Homogeneous [%]", "Heterogeneous
            [%]", "Highly heterogeneous [%]"]
            summary_df[numeric_cols] = summary_df[numeric_cols].round(0).astype(int)

            # Store DataFrame
            summary_dfs[f"TP_{tp}"] = summary_df

        return summary_dfs

summary_results = create_tp_summary_percentage_df(results_df)
summary_results

"""## Plotting"""

import pandas as pd
import plotly.express as px
!pip install ipdb
import numpy as np
from collections import defaultdict

"""https://plotly.com/python/sunburst-charts/

Creating the structure df for sunburst chart
"""

rts_datasets = {   # collection of RTSs per area
    "T1": rtsT1_v2,
    "T2": rtsT2_v2,
    "T3": rtsT3_v2,
    "T4": rtsT4_v2,
    "T6": rtsT6_v2
}
# Table of propability of TPs per area
rts_condensed = defaultdict(lambda: defaultdict(int))
Terraintypes = np.arange(5)+1
Regions = ["T1","T2","T3","T4","T6"]
TP_lookup = ["TP_1","TP_2","TP_3","TP_4","TP_6"]
ClassIndex = np.arange(4)
Classifiers = ["Highly homogeneous [%]","Homogeneous [%]","Heterogeneous [%]","Highly
heterogeneous [%]"]

for key in Regions: # key = region
  for tptype in [1,2,3,4,6]: # ttype = art
    rts_condensed[key][tptype] = np.sum(np.array(rts_datasets[key]["TP"])==tptype)
```

```
# Mapping of dataset names (T1, T2, etc.) to their corresponding Region
region_mapping = {
    "T1": "Southern Taymyr",
    "T2": "Northern Olenek",
    "T3": "Chokurdakh",
    "T4": "Iultinsky (Chukotka)",
    "T6": "Southern Verkhoyansk Range"
}

# Mapping Terrain Position Names
terrain_mapping = {
    "TP_1": "Sea",
    "TP_2": "Lake",
    "TP_3": "River",
    "TP_4": "Gully",
    "TP_5": "Others",
    "TP_6": "Ponds + Gully"
}

summary_results # Data frame sorted by TP (+ area and similarity classification)

# Goal: Region | Terrain Position | Similarity Classification | Classification Number (for colour scheme)| Counts

Results = pd.DataFrame(columns=['Region','Terrain Position','Classification', 'ClassificationNumber','Count', "S.Percentage"])

# generate a vector of coordinates for areas and TPs
grid1, grid2, grid3 = np.meshgrid(Regions, Terraintypes, ClassIndex)
Stepvector = np.array(list(zip(grid1.ravel(), grid2.ravel(), grid3.ravel())))

# Results Dataframe line by line
for ll in np.arange(Stepvector.shape[0]):
  tmp_Region = region_mapping[Stepvector[ll,0]]
  tmp_TP = terrain_mapping[TP_lookup[int(Stepvector[ll,1])-1]]
  tmp_Class = Classifiers[int(Stepvector[ll,2])]
  tmp_total_Count = rts_condensed[Stepvector[ll,0]][int(Stepvector[ll,1])]

  # Add statistical results (similarity classification)
  tmp_table = summary_results[TP_lookup[int(Stepvector[ll,1])-1]]
  tmp_table2 = tmp_table[tmp_table['Area'] == tmp_Region]

  tmp_Class_num = int(Stepvector[ll,2])+1

  if tmp_table2[tmp_Class].empty:
    tmp_Count = 0  # null slumps is default
    tmp_class_value = 0
  else:
    tmp_class_value = tmp_table2[tmp_Class].values[0]
    # normalize with totaltpcounts
    tmp_Count = np.round(tmp_class_value*tmp_total_Count/100,2)

  Results.loc[ll] = pd.Series({'Region':tmp_Region, 'Terrain Position':tmp_TP, 'Classification':tmp_Class,'ClassificationNumber':tmp_Class_num, 'Count':tmp_Count, "S.Percentage":tmp_class_value})

Results

# Remove rows with Count == 0
TP_SS_Results = Results[Results['Count'] != 0]

# Adding the values for TP_6 that the algorithim could not finde by hand
TP_SS_Results.loc[80] = ['Iultinsky (Chukotka)', 'Ponds + Gully', "Heterogeneous [%]", 3, (np.round(12*11/100,2)), 12]
TP_SS_Results.loc[81] = ['Iultinsky (Chukotka)', 'Ponds + Gully', "Homogeneous [%]", 2, (np.round(70*11/100,2)), 70]
TP_SS_Results.loc[82] = ['Iultinsky (Chukotka)', 'Ponds + Gully', "Highly homogeneous [%]", 1, (np.round(18*11/100,2)), 18]

# Adding the values for single slumps that the algorithim could not finde by hand
TP_SS_Results.loc[83] = ['Southern Taymyr', 'River', "Highly heterogeneous [%]", 4, 1.00, 0]
```

```
TP_SS_Results.loc[84] = ['Southern Verkhoyansk Range', 'Gully', "Highly heterogeneous [%]", 4, 1.00, 0]

#Define ClassificationNumber as integer for color sheme
TP_SS_Results['ClassificationNumber'] = TP_SS_Results['ClassificationNumber'].astype(int)
TP_SS_Results

# save the data
from google.colab import drive
import pandas as pd
#drive.mount('/content/drive')

#TP_SS_Results.to_csv('/content/drive/My Drive/Colab Notebooks/Data/TP_SS_Results.csv' )

# load the data
#TP_SS_Results = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Data/TP_SS_Results.csv')
#TP_SS_Results

# create sunburst chart
import plotly.express as px
import numpy as np

# Custom continuous color scale with specific colors
custom_continuous_colors = [
    [0, "gold"],  # Lower end of the scale (smallest values)
    [0.33, "lightyellow"], # Midpoint of the scale
    [0.66, "lavender"], # Further midpoint
    [1, "darkviolet"]         # Upper end of the scale (largest values)
]

fig = px.sunburst(TP_SS_Results,
                  path=['Region', 'Terrain Position', 'Classification'],
                  values='Count',
                  color='ClassificationNumber',
                  color_continuous_scale=custom_continuous_colors,
                  )
fig.update_layout(
                  showlegend=True,
                  coloraxis=dict(colorscale=custom_continuous_colors),
                  coloraxis_colorbar=dict(
                      tickvals=[4, 3, 2, 1],  # Define the tick values for the colorbar
                      ticktext=["Highly heterogeneous", "Heterogeneous", "Homogeneous", "Highly homogeneous"]
                  )
)
fig.show()

"""Error calculation"""

TP_SS_Error = TP_SS_Results
# table with error calculations
TP_SS_Error["S.Percentage Error"] = 0.5
TP_SS_Error["Count Error"] = 0  # Initialize 'Count Error' column to 0

# Use a loop or apply to calculate "Count Error" based on conditions
for index in TP_SS_Error.index:
    if TP_SS_Error.loc[index, "S.Percentage"] != 0:  # Check for non-zero percentage
        TP_SS_Error.loc[index, "Count Error"] = np.round(np.sqrt((TP_SS_Error.loc[index, "S.Percentage Error"] / TP_SS_Error.loc[index, "S.Percentage"]) ** 2) * TP_SS_Error.loc[index, "Count"],2)

# Select required columns
TP_SS_Error = TP_SS_Error[["Region", "Terrain Position", "Classification", "S.Percentage", "S.Percentage Error", "Count", "Count Error"]]

TP_SS_Error

"""# Morphology, spectral slope similarity compairison (sub-hypothesis 4)
```

```
## Calcutaion of statistics for all slumps
"""

from scipy.stats import shapiro
!pip install scikit-posthocs
# during analysis the version "0.11.2-py3-none-any.whl.metadata (5.8 kB)" was used.
#That version is no longer availabe and version 0.11.3 can produce slightly different
results.
import scipy.stats as stats
import scikit_posthocs as sp
import pandas as pd

### Dunn's Test and Kruskal-Wallis tests

# Define numerical columns to compare
columns = ["TCB_slope", "TCG_slope", "TCW_slope"]

comparison_results = {}

# Group GDFs by T number (Area) and Shape value
for T_key, sub_gdfs in gdf_singleRTSs_plus_Property.items():
    shape_groups = {}  # Dictionary to store groups by Shape value

    # Organize GDFs by Shape value
    for df_name, df in sub_gdfs.items():
        parts = df_name.split("_")
        Shape_value = int(parts[3][5:])  # Extract Shape value from 'ShapeX'

        if Shape_value not in shape_groups:
            shape_groups[Shape_value] = []

        shape_groups[Shape_value].append(df)

    # Perform statistical comparisons within each Shape group
    for Shape_value, gdf_list in shape_groups.items():
        if len(gdf_list) > 1:  # Only compare if we have multiple GDFs
            comparison_results[f"T{T_key}_Shape{Shape_value}"] = {}

            for col in columns:
                # Combine data from all GDFs in this group
                combined_data = [gdf[col].dropna().values for gdf in gdf_list]

                # Kruskal-Wallis test (checks if there's any difference)
                H_stat, p_kw = stats.kruskal(*combined_data)

                if p_kw < 0.05:  # If significant, perform Dunn's test
                    dunn_results = sp.posthoc_dunn(combined_data, p_adjust=
                    "bonferroni")
                    comparison_results[f"T{T_key}_Shape{Shape_value}"][col] = {
                    "Kruskal_p": p_kw, "Dunn": dunn_results}
                else:
                    comparison_results[f"T{T_key}_Shape{Shape_value}"][col] = {
                    "Kruskal_p": p_kw, "Dunn": None}

# Print results
for group, results in comparison_results.items():
    print(f"\n📊 Statistical Comparisons for {group}:")
    for col, stats_dict in results.items():
        print(f"  🔹 {col}: Kruskal-Wallis p-value = {stats_dict['Kruskal_p']:.4f}")
        if stats_dict["Dunn"] is not None:
            print(f"     Dunn's test results:\n{stats_dict['Dunn']}")
        else:
            print("     No significant differences found.")

import matplotlib.pyplot as plt
from matplotlib.table import Table

results = []

# Iterate over each group (T, Shape)
for group, group_results in comparison_results.items():
```

```
        # Extract T and Shape from the group string
        T_value, Shape_value = group.split('_')[0][1:], group.split('_')[1][5:]

        # Process each column in the group's results
        for slope_key, stats_dict in group_results.items():
            # If Dunn's test is available
            if stats_dict['Dunn'] is not None:
                matrix = stats_dict['Dunn']

                # Iterate over the rows of the matrix
                for row_index, row_values in matrix.iterrows():
                    total_columns = len(row_values) - 1  # Subtracting 1 to exclude the comparison with itself
                    exceed_count = (row_values > 0.05).sum() - 1  # Count values exceeding 0.05
                    percentage = (exceed_count / total_columns) * 100

                    # Append the result to the results list
                    results.append({
                        "Area": T_value,  # Extracted T value
                        "Shape": Shape_value,  # Extracted TP value
                        "Slope": slope_key,  # The slope key (e.g., 'TCB_slope')
                        "RTSIndex": row_index,  # The row index (single slump index)
                        "TotalColumns": total_columns,  # Total columns in the matrix (excluding the diagonal)
                        "Count>0.05": exceed_count,  # Count of values exceeding 0.05
                        "Percentage>0.05": percentage  # Percentage of values exceeding 0.05
                    })

    # Print the results
    results_df = pd.DataFrame(results)
    print(results_df)

    """## Homogenitiy classification"""

    ## create tables from df, cecking if patterns are visible
    # Function to get row colors based on Percentage
    def get_row_color(percentage):
        if percentage == 0:
            return "darkviolet"
        elif percentage < 50:
            return "lavender"
        elif percentage >= 50 and percentage < 100:
            return "lightyellow"
        elif percentage == 100:
            return "gold"
        return "white"

    # Function to create a table in a specific subplot
    def create_styled_table_in_subplot(ax, df, title="Table"):
        ax.axis("off")
        ax.set_title(title, fontsize=16, pad=26)

        table = Table(ax, bbox=[0, 0, 1, 1])
        nrows, ncols = df.shape

        col_labels = df.columns
        for col_idx, label in enumerate(col_labels):
            table.add_cell(-1, col_idx, text=label, width=1, height=0.2, facecolor="lightgray", loc="center")

        # Row cells
        prev_subarea = None
        for row_idx, row in df.iterrows():
            current_subarea = row["Area"]
            edgecolor = "black"
            if prev_subarea != current_subarea:
                edgecolor = "black"

            prev_subarea = current_subarea
```

```
            for col_idx, value in enumerate(row):
                # Get cell color
                if col_labels[col_idx] == "Percentage>0.05":
                    cell_color = get_row_color(row["Percentage>0.05"])
                else:
                    cell_color = "white"

                table.add_cell(
                    row_idx,
                    col_idx,
                    text=str(value),
                    width=1,
                    height=0.2,
                    facecolor=cell_color,
                    loc="center",
                    edgecolor=edgecolor,
                )

        # Add the table to the subplot
        ax.add_table(table)

    # Prepare the DataFrame subsets and clean up slope names
    results_df_cleaned = results_df.drop(columns=["Slope"])  # Remove the Slope column
    unique_slopes = results_df["Slope"].unique()

    # Create subplots for the tables
    fig, axes = plt.subplots(1, len(unique_slopes), figsize=(54, 40))  # Increased size for higher resolution
    fig.tight_layout(pad=5)

    # Create a table for each slope type
    for ax, slope in zip(axes, unique_slopes):
        slope_df = results_df[results_df["Slope"] == slope].drop(columns=["Slope"])
        clean_title = f"Slope: {slope.replace('_slope', '')}"  # Clean slope name
        create_styled_table_in_subplot(ax, slope_df, title=clean_title)

    plt.show()

    def create_tp_summary_percentage_df(results_df):
        summary_dfs = {}  # Dictionary to store summary DataFrames for each Shape

        # Get unique Shape values
        Shape_values = results_df["Shape"].unique()

        for shape in Shape_values:
            # Filter for current Shape
            shape_df = results_df[results_df["Shape"] == shape]

            summary_data = []
            areas = shape_df["Area"].unique()

            for area in areas:
                area_df = shape_df[shape_df["Area"] == area]
                area_total = len(area_df)

                summary_data.append([
                    f"{area}",
                    (area_df["Percentage>0.05"] == 0).sum() / area_total * 100,  # Highly heterogeneous
                    ((area_df["Percentage>0.05"] > 0) & (area_df["Percentage>0.05"] < 50)).sum() / area_total * 100,  # Heterogeneous
                    ((area_df["Percentage>0.05"] >= 50) & (area_df["Percentage>0.05"] < 100)).sum() / area_total * 100,  # Homogeneous
                    (area_df["Percentage>0.05"] == 100).sum() / area_total * 100  # Highly homogeneous
                ])

            # Create DataFrame
            summary_df = pd.DataFrame(summary_data, columns=[
                "Area",
                "Highly heterogeneous [%]",
                "Heterogeneous [%]",
```

```
            "Homogeneous [%]",
            "Highly homogeneous [%]"
        ])

        area_mapping = {
            "T1": "Southern Taymyr",
            "T2": "Northern Olenek",
            "T3": "Chokurdakh",
            "T4": "Iultinsky (Chukotka)",
            "T6": "Southern Verkhoyansk Range"
        }

        summary_df["Area"] = summary_df["Area"].replace(area_mapping)

        # Round numeric columns
        numeric_cols = ["Highly homogeneous [%]", "Homogeneous [%]", "Heterogeneous [%]", "Highly heterogeneous [%]"]
        summary_df[numeric_cols] = summary_df[numeric_cols].round(0).astype(int)

        # Store DataFrame
        summary_dfs[f"Shape_{shape}"] = summary_df

    return summary_dfs

summary_results = create_tp_summary_percentage_df(results_df)
summary_results

"""## Plotting"""

import pandas as pd
import plotly.express as px
!pip install ipdb
import numpy as np
from collections import defaultdict

rts_datasets = {
    "T1": rtsT1_v2,
    "T2": rtsT2_v2,
    "T3": rtsT3_v2,
    "T4": rtsT4_v2,
    "T6": rtsT6_v2
}

rts_condensed = defaultdict(lambda: defaultdict(int))
Shapetypes = np.arange(2)+1
Regions = ["T1","T2","T3","T4","T6"]
Shape_lookup = ["Shape_0", "Shape_1", "Shape_2"]
ClassIndex = np.arange(4)
Classifiers = ["Highly homogeneous [%]","Homogeneous [%]","Heterogeneous [%]","Highly heterogeneous [%]"]

for key in Regions: # key = region
  for shtype in [1,2,3,4,6]: # ttype = art
    rts_condensed[key][shtype] = np.sum(np.array(rts_datasets[key]["Shpe"])==shtype)

# Mapping of dataset names (T1, T2, etc.) to their corresponding Region
region_mapping = {
    "T1": "Southern Taymyr",
    "T2": "Northern Olenek",
    "T3": "Chokurdakh",
    "T4": "Iultinsky (Chukotka)",
    "T6": "Southern Verkhoyansk Range"
}

# Mapping Morphology Names
Shape_mapping = {
    "Shape_0": "Thermocirque",
    "Shape_1": "Thermoterrace",
    "Shape_2": "Combination"
}

summary_results # Data frame sorted by shape class/morphology ("Shape_*")
```

```
2978
2979
2980  Results = pd.DataFrame(columns=['Region','Morphology','Classification',
      'ClassificationNumber','Count', "S.Percentage"])
2981
2982  # generate a vector of coordinates for areas and Shapes
2983  grid1, grid2, grid3 = np.meshgrid(Regions, Shapetypes, ClassIndex)
2984  Stepvector = np.array(list(zip(grid1.ravel(), grid2.ravel(), grid3.ravel())))
2985
2986  # Results Dataframe line by line
2987  for ll in np.arange(Stepvector.shape[0]):
2988    tmp_Region = region_mapping[Stepvector[ll,0]]
2989    tmp_Shape = Shape_mapping[Shape_lookup[int(Stepvector[ll,1])]]#-1!!!!!!
2990    tmp_Class = Classifiers[int(Stepvector[ll,2])]
2991    tmp_total_Count = rts_condensed[Stepvector[ll,0]][int(Stepvector[ll,1])]
2992
2993    tmp_table = summary_results[Shape_lookup[int(Stepvector[ll,1])]]#-1!!!!!!!!!
2994    tmp_table2 = tmp_table[tmp_table['Area'] == tmp_Region]
2995
2996    tmp_Class_num = int(Stepvector[ll,2])+1
2997
2998    if tmp_table2[tmp_Class].empty:
2999      tmp_Count = 0
3000    else:
3001      tmp_class_value = tmp_table2[tmp_Class].values[0]
3002
3003      tmp_Count = np.round(tmp_class_value*tmp_total_Count/100,2)
3004
3005    Results.loc[ll] = pd.Series({'Region':tmp_Region, 'Morphology':tmp_Shape,
      'Classification':tmp_Class,'ClassificationNumber':tmp_Class_num, 'Count':tmp_Count,
      "S.Percentage": tmp_class_value})
3006
3007  Results
3008
3009  # Delet if Count == 0
3010  Shape_SS_Results = Results[Results['Count'] != 0]
3011
3012  # Adding the values for TP_6 that the algorithim could not finde by hand
3013  Shape_SS_Results.loc[50] = ['Southern Taymyr', 'Thermocirque', "Highly heterogeneous
      [%]", 4, (np.round(2*14/100,2)), 2]
3014  Shape_SS_Results.loc[51] = ['Southern Taymyr', 'Thermocirque', "Heterogeneous [%]", 3,
       (np.round(33*14/100,2)), 33]
3015  Shape_SS_Results.loc[52] = ['Southern Taymyr', 'Thermocirque', "Homogeneous [%]", 2, (
      np.round(62*14/100,2)), 62]
3016  Shape_SS_Results.loc[53] = ['Southern Taymyr', 'Thermocirque', "Highly homogeneous
      [%]", 1, (np.round(2*14/100,2)), 2]
3017
3018  Shape_SS_Results.loc[54] = ['Northern Olenek', 'Thermocirque', "Highly heterogeneous
      [%]", 4, (np.round(44*3/100,2)), 44]
3019  Shape_SS_Results.loc[55] = ['Northern Olenek', 'Thermocirque', "Homogeneous [%]", 2, (
      np.round(44*3/100,2)), 44]
3020  Shape_SS_Results.loc[56] = ['Northern Olenek', 'Thermocirque', "Highly homogeneous
      [%]", 1, (np.round(11*3/100,2)), 11]
3021
3022  Shape_SS_Results.loc[57] = ['Chokurdakh', 'Thermocirque', "Heterogeneous [%]", 3, (np.
      round(17*10/100,2)), 17]
3023  Shape_SS_Results.loc[58] = ['Chokurdakh', 'Thermocirque', "Homogeneous [%]", 2, (np.
      round(57*10/100,2)), 57]
3024  Shape_SS_Results.loc[59] = ['Chokurdakh', 'Thermocirque', "Highly homogeneous [%]", 1,
       (np.round(27*10/100,2)), 27]
3025
3026  Shape_SS_Results.loc[60] = ['Iultinsky (Chukotka)', 'Thermocirque', "Heterogeneous
      [%]", 3, (np.round(17*22/100,2)), 17]
3027  Shape_SS_Results.loc[61] = ['Iultinsky (Chukotka)', 'Thermocirque', "Homogeneous [%]",
       2, (np.round(73*22/100,2)), 73]
3028  Shape_SS_Results.loc[62] = ['Iultinsky (Chukotka)', 'Thermocirque', "Highly
      homogeneous [%]", 1, (np.round(11*22/100,2)), 11]
3029
3030  Shape_SS_Results.loc[63] = ['Southern Verkhoyansk Range', 'Thermocirque', "Highly
      heterogeneous [%]", 4, (np.round(11*6/100,2)), 11]
3031  Shape_SS_Results.loc[64] = ['Southern Verkhoyansk Range', 'Thermocirque',
      "Heterogeneous [%]", 3, (np.round(67*6/100,2)), 67]
```

```
Shape_SS_Results.loc[65] = ['Southern Verkhoyansk Range', 'Thermocirque', "Homogeneous [%]", 2, (np.round(22*6/100,2)), 22]

# Adding the values for single slumps that the algorithim could not finde by hand
Shape_SS_Results.loc[83] = ['Southern Verkhoyansk Range', 'Thermoterrace', "Highly heterogeneous [%]", 4, 1.00, 0]

#Define ClassificationNumber as integer for color scheme
Shape_SS_Results['ClassificationNumber'] = Shape_SS_Results['ClassificationNumber'].astype(int)
Shape_SS_Results

# Save the data
from google.colab import drive
import pandas as pd
#drive.mount('/content/drive')

#TP_SS_Results.to_csv('/content/drive/My Drive/Colab Notebooks/Data/TP_SS_Results.csv' )

#load the data
#TP_SS_Results = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Data/TP_SS_Results.csv')
#TP_SS_Results

# create sunburst chart
# Continuous color scale with specific colors
custom_continuous_colors = [
    [0, "gold"],  # Lower end of the scale (smallest values)
    [0.33, "lightyellow"], # Midpoint of the scale
    [0.66, "lavender"], # Further midpoint
    [1, "darkviolet"]         # Upper end of the scale (largest values)
]
fig = px.sunburst(Shape_SS_Results,
                  path=['Region', 'Morphology', 'Classification'],
                  values='Count',
                  color='ClassificationNumber',
                  color_continuous_scale=custom_continuous_colors,
                  )
fig.update_layout(
                  showlegend=True,
                  coloraxis=dict(colorscale=custom_continuous_colors),
                  coloraxis_colorbar=dict(
                      tickvals=[4, 3, 2, 1],  # Define the tick values for the colorbar
                      ticktext=["Highly heterogeneous", "Heterogeneous", "Homogeneous", "Highly homogeneous"]
                  )
)
fig.show()

"""Error calculation"""

Shape_SS_Error = Shape_SS_Results
# table with error calculations
Shape_SS_Error["S.Percentage Error"] = 0.5
Shape_SS_Error["Count Error"] = 0  # Initialize 'Count Error' column to 0

# Use a loop or apply to calculate "Count Error" based on conditions
for index in Shape_SS_Error.index:
    if Shape_SS_Error.loc[index, "S.Percentage"] != 0:  # Check for non-zero percentage
        Shape_SS_Error.loc[index, "Count Error"] = np.round(np.sqrt((Shape_SS_Error.loc[index, "S.Percentage Error"] / Shape_SS_Error.loc[index, "S.Percentage"]) ** 2) * Shape_SS_Error.loc[index, "Count"],2)

# Select required columns
Shape_SS_Error = Shape_SS_Error[["Region", "Morphology", "Classification", "S.Percentage", "S.Percentage Error", "Count", "Count Error"]]

Shape_SS_Error
```

```
"""# Relation of morphologie and terrain positions


"""

import pandas as pd
from collections import Counter
import plotly.express as px
import numpy as np

from shapely.geometry import MultiPoint, Polygon
import pandas as pd
from collections import Counter
## Data with center coordinates of Regions for Zenodo
# Region Name Mapping & Corresponding FeatureCollection IDs
region_mapping = {
    "T1": "Southern Taymyr",
    "T2": "Northern Olenek",
    "T3": "Chokurdakh",
    "T4": "Iultinsky (Chukotka)",
    "T6": "Southern Verkhoyansk Range"
}

region_feature_collections = {
    "T1": "projects/ee-moritzjulia7/assets/Area_T1",
    "T2": "projects/ee-moritzjulia7/assets/Area_T2",
    "T3": "projects/ee-moritzjulia7/assets/Area_T3",
    "T4": "projects/ee-moritzjulia7/assets/Area_T4",
    "T6": "projects/ee-moritzjulia7/assets/Area_T6"
}

# Morphology Name Mapping
Shape_mapping = {
    "Shape0": "Thermocirque",
    "Shape1": "Thermoterrace",
    "Shape2": "Combination"
}

# Terrain Position Mapping
terrain_mapping = {
    "TP1": "Sea",
    "TP2": "Lake",
    "TP3": "River",
    "TP4": "Gully",
    "TP5": "Others",
    "TP6": "Ponds + Gully"
}

# Compute centroid for each region's FeatureCollection
centroid_mapping = {}

for region, asset_id in region_feature_collections.items():
    feature_collection = ee.FeatureCollection(asset_id)  # Load FeatureCollection
    from GEE
    region_feature = feature_collection.first()  # Get the first (and only) feature
    coordinates = region_feature.geometry().coordinates().getInfo()[0]  # Extract
    polygon coordinates

    # Convert coordinates to Shapely Polygon and compute centroid
    polygon = Polygon(coordinates)
    centroid = polygon.centroid
    centroid_mapping[region_mapping[region]] = (centroid.x, centroid.y)

# Flatten dictionary and collect all df names
df_keys = [df_key for sub_dict in gdf_singleRTSs_plus_Property.values() for df_key in
sub_dict.keys()]

# Parse and clean keys
parsed_keys = []
region_terrain_counts = Counter()

for df_key in df_keys:
```

```
        parts = df_key.split("_")
        region, terrain_position, morphology = parts[0], parts[2], parts[3]  # Ignore `fid`

        # Apply mappings
        region_name = region_mapping.get(region, region)  # Map region, fallback to original if missing
        terrain_name = terrain_mapping.get(terrain_position, terrain_position)  # Map terrain, fallback if missing
        morphology_name = Shape_mapping.get(morphology, morphology)  # Map morphology, fallback if missing

        parsed_keys.append((region_name, terrain_name, morphology_name))
        region_terrain_counts[(region_name, terrain_name)] += 1  # Count (Region, Terrain Position) pairs

    # Count occurrences of unique (Region, Terrain Position, Morphology)
    df_count = Counter(parsed_keys)

    # Create DataFrame
    data = []
    for (region_name, terrain_name, morphology_name), count in df_count.items():
        total_count = region_terrain_counts[(region_name, terrain_name)]  # Get count for (Region, Terrain Position)
        row_percentage = (count / total_count) * 100  # Compute percentage
        centroid_coords = centroid_mapping.get(region_name, (None, None))  # Fetch centroid

        data.append([region_name, terrain_name, morphology_name, count, centroid_coords])

    df_result = pd.DataFrame(data, columns=["Region", "Terrain Position", "Morphology", "Count", "Region Center Coordinates"])

    # Display result
    print(df_result)

    df_result.to_csv('Data_RTS_Morphology-TP_Siberia.csv', index=False)  # Save CSV without the index

    #create sunburst chart
    fig = px.sunburst(df_result,
                      path=['Region', 'Terrain Position', 'Morphology'],
                      values='Count',
                      color="Region",  # Coloring by Region
                      color_discrete_map={
                          "Southern Taymyr": "lightblue",
                          "Northern Olenek": "blue",
                          "Chokurdakh": "orange",
                          "Iultinsky (Chukotka)": "#734F96",
                          "Southern Verkhoyansk Range": "darkred"})

    fig.update_layout(
                      showlegend=True
    )
    fig.show()
    #fig.write_html("/content/drive/My Drive/Colab Notebooks/Data/sunburst_Morphology-TP.html")
```