Project 1: Counting Melted pixels in Greenland for 2007 using Python

Introduction:

The intention of this project was to analyze imagery showing melt in Greenland, This was generated in a previous course on remote sensing. 365 TIF images contained within the project folder are of Greenland. The images have had a ratio applied allowing quantification of Greenland's 2007 melted area. This data is used by climate scientists in quantifying melt and predicting sea level rise in response to anthropogenic climate change. Pre-processing raw satellite imagery in ASCII format and converting into TIF files was done prior to this project. Images are from the Scanning Multi-channel Microwave Radiometer (SMMR) and the Special Sensor Microwave/Imager (SSM/I) instruments onboard a defense meteorological satellite, DMSR mission. Pre-processing combined two microwave images with differing polarities of each day. The XPGR ratio was then applied to each combined image. The XPGR ratio acts as a binary indicator of the state of melt found within each pixel on each observed day, by creating a threshold that is a proxy indicator of snow wetness. If a pixel's XPGR-corrected brightness value is above the threshold, the pixel has released energy as the ice in the image changes state from solid to liquid. Each pixel represents an area of 25km^2. Each of the 365 TIF images are 60 x 109 pixels in size.

There is uncertainty in this dataset. Errors in brightness values for each file representing an observable day may be present. Errors were largely removed in pre-processing, most containing pixels missing values. Melted pixel counts will be different than those determined by the actual research by the NSIDC because a mask was not applied to remove coastal pixels that melt and refreeze frequently and are very variable. This project does not seek to correct these uncertainties, nor does it express an accurate melt extent of the Greenland Ice sheet. Although it's accuracy is in question, these methods produced more precise accounts than did my initial analysis using ArcGIS, which repeat-counted melted pixels, generating a highly inaccurate melt area that was too large.

Methods - What the code does and how to run it:

The project folder contains python code, output text, imagery needed for the project, and the whitepaper folder includes screenshots of progress, **requirements text of modules needed**, and this paper. A dictionary, *months*, is set up that will be populated with pixels above a threshold value of -0.0158. Pixels with values greater than this are considered to be melting, pixels under this value are not counted. Overall, the code only counts pixels that have melted, and only counts them once. These are stored into a dictionary where each month (Jan, Feb, etc) is a key that calls forth a summed count of pixels for each of the daily TIF images. These are totaled, producing all pixels showing ice melt in Greenland for 2007. The area is determined. User prompts allow the user to explore these values within the CIL. When running through the 365 files, a progress bar was created to show the user the code is processing the files.

Lines 14-24: Module imports including error checking prompting user to download necessary modules. Directions are included within Requirements.txt inside Whitepaper folder.

Lines 26-27: Global variables, dictionary {months} is created, and has_run allows to see when images have been processed.

Lines 29-56: main() function. User input is checked for validity in lines 36-54.

Lines 58-157: This is the function process(). This is the majority of the program, where the dictionary is populated and the images are opened and closed, and the count of pixels within each image that are above the threshold is applied. Local variables are set within this, including numerical variable threshold, the current working directory is set, melt variable storing all melted pixels, variables for each month, and fnum which allows the progress bar to be created showing when the process() function is opening each TIF image and running the program. The for loop grub (lines 74-76) is for counting iterations in creation of max loading value for the progress loading bar. A for loop (Lines 83-125) is used to split the pathname of each file and count these these as days, which are then opened, pixels above the melt threshold are counted, and if the pixel has melted already, it is not counted. Each time a day TIF file is opened, the count is reset to zero, and any new melted pixel is stored into the month that day is located in. This takes many if:elif statements. The final else statement says to run function path_err() to return an error to the user (defined in Lines 163-165) if there is an issue, and to quit the program. Lines 126-128 process the loading bar, where bar.next() increments the loading bar, and bar.finish() stops the loading bar when all melted pixels have been counted in the full 365 images. Line 129 totals all melt into a final sum for the year. Lines 131-143 populate the {month} dictionary for each month. Lines 145-150 force specific global variables in this local function to be global. This generates a warning output when the program is run, which is disregarded. These are being forced so that has_run variable can indicate the images have been processed, making it boolean by setting it to 1 (true = processed images). Global area allows function melt_area() (Defined in 159-161) to populate the dictionary and output in the CLI with the total melted pixel count in km^2. The final it-elif-else command allows for error checking and prints the total melted pixels, else months if those are provided as input by the user.

Lines 159-161: The melt_area(): The total melt area function, to be populated in process() into the dictionary and inputted both to .txt and CLI.

Lines163-165: The path_err() function allows for an error to occur if a TIF file does not exist.

Lines 167-211: The log_output() function allows for the output file to be created. The file is ‘a’ appended, not written, so that a running log can show how the program has changed. This program will be used in the future, and keeping an appended log is good for seeing what changes do to output. the log.txt is included in the Project folder. The if statement is to allow the appending to occur up to a given size. Lines 176-210 are for formatting the .txt file.

Line 213: main() function is executed, and the story begins!