## Test and run issues

To make my program work, I wrote Python unit test scripts to test the functions in my\_modules.calculate and my\_modules.io, using the TestCase class in unittest. The setUp() method was used in order to create a sample raster data before testing. The test\_ method is then used to test the respective functions, where I tested a total of four functions: test\_read\_raster\_data, test\_multiply\_raster, test\_add\_rasters, and test\_rescale\_raster. running the script will execute all the test methods . The final test results for each test method are the same as the actual results. Below is my test code.

###### Test code:

import unittest

import my\_modules.calculate as cal

import my\_modules.io as io

class TestRasterFunctions(unittest.TestCase):

def setUp(self):

# Create the sample raster data in the test case

self.raster = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

def test\_read\_raster\_data(self):

# Test function for reading raster data

file\_path = "Datas/Geology\_TEST.txt"

with open(file\_path, "w") as f:

f.write("0,1,2\n3,4,5\n6,7,8\n")

expected\_result = self.raster

actual\_result = io.read\_raster\_data(file\_path)

self.assertEqual(expected\_result, actual\_result)

def test\_multiply\_raster(self):

# Test the function that multiplies the raster data by the weighting factor

weight = 2

expected\_result = [[0, 2, 4], [6, 8, 10], [12, 14, 16]]

actual\_result = cal.multiply\_raster(self.raster, weight)

self.assertEqual(expected\_result, actual\_result)

def test\_add\_rasters(self):

# Test the function that adds raster data

raster1 = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

raster2 = [[1, 1, 1], [1, 1, 1], [1, 1, 1]]

raster3 = [[2, 2, 2], [2, 2, 2], [2, 2, 2]]

expected\_result = [[3, 3, 3], [3, 3, 3], [3, 3, 3]]

actual\_result = cal.add\_rasters(raster1, raster2, raster3)

self.assertEqual(expected\_result, actual\_result)

def test\_rescale\_raster(self):

# Test function to rescale raster data

new\_min = 0

new\_max = 255

expected\_result = [[0, 31, 63], [95, 127, 159], [191, 223, 255]]

actual\_result = cal.rescale\_raster(self.raster, new\_min, new\_max)

self.assertEqual(expected\_result, actual\_result)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

###### Problems when testing code:

I have encountered various problems in completing the testing and development of the software, some of which have been resolved and others are still being worked on. Below is a description of the problems and how I have solved them.

Problem 1: Lists not loading correctly

Since at the beginning in site.py I fixed the values of n\_row and n\_cols when defining the add\_rasters function, the code is as follows:

def add\_rasters(raster1, raster2, raster3):

n\_row=530

n\_cols=335

added\_raster=[];

for i in range(n\_row): added\_row=[];

for j in range(n\_cols): addraster = raster1[i][j] + raster2[i][j] + raster3[i][j]

Solution:

As a result it was not possible to read and write raster data properly when testing the code, which affected the flexibility of the program. By thinking about it, I decided to write the code in a different way, eliminating fixed values and reading the raster's row values directly. The code is as follows:

 n\_row=len(raster1)

 n\_cols=len(raster1[0])

With this change, the test code was finally able to continue down the line.

Problem 2: Confusing functions with no logic

As my code file was written very confusingly at the beginning, the main functions were mixed up with the calculation functions, causing me to load the GUI interface directly when testing without being able to know the detailed test results in it.

Solution:

I adjusted the structure of my code and divided it into three parts: site.py, io.py and calculate.py. In Test function.py only read\_raster\_data, multiply\_raster, add\_rasters, rescale\_raster are included in the part.

So far my test code can run normally and "ok" appears in the console.

###### Problems in developing the program:

Problem 1: PIL version compatibility issues

Before I started writing the program, I ran into a tricky problem where I had installed the Pillow version 9.5.0, but the Imaging extension was for version 9.4.0 of Pillow, so the two library versions were not compatible, resulting in the Imaging extension not working properly.

The error codes are as follows：ImportError: The imaging extension was built for another version of Pillow or PIL:Core version: 9.4.0Pillow version: 9.5.0.

Solution:  
First, check what version of my program and all related libraries are.  
  
I then found that I had two versions of pillow on my computer, which may have been one of the reasons why I was not loading correctly, so I uninstalled both versions in CMD and reinstalled the matching version of pillow.The steps were as follows:  
  
Type ----pip uninstall Pillow in the console to uninstall the current version of pillow.  
Type ----pip install Pillow== to select the version to install and update the version of Pillow to match it. Type ----pip install --upgrade Pillow to update pillow.

Problem 2: Sum limit of sliders

When writing the code for the slider, simply setting the length of the slider and the size of the position ignores the nature of the operation in this program. As this is a weighted slider, the three important factors form a whole and therefore the sum of their respective shares should always remain 1.

Solution:

In order to solve this problem, it occurred to me that I could define the total value of the three sliders by setting them. I added a line total = geology\_val + transport\_val + population\_val code to define their sum.

And two if conditional statements have been added, setting the slider to be resized by a scaling factor if total > 1/if total < 1 if the total value adds up to more or less than 1 scale\_factor = 1 / total so that it always sums to 1.

And I think another advantage of writing the code this way is that sometimes people tend to ignore the sum of the weights, so the automatic adjustment is extra humane and avoids unnecessary low-level errors.

Problem 3: Unable to save weighted images individually

When saving the generated images, it is not possible to save the Weighted Raster images individually, but rather the four plots are saved together as a whole. Initially, I used the plt.subplot() function to create four subplots in the graphics window, and I suspect that I was unable to output an image of one of them individually because all four plots were in one canvas. I then changed to using the ax.imshow() method, the code is as follows:

axs[3].imshow(rescaled\_raster)

axs[3].set\_title('Weighted Raster')

Trying to adjust the parameters for each image individually still does not change the number of images output.

Solution:

I found the problem was that I had forgotten to write the plt.close() function, which caused the image saved each time an image was created using matplotlib to contain both the current and all previous images. The changed code is as follows:

 plt.close()# Close any previously open windows

 plt.imshow(rescaled\_raster)

 plt.title('Weighted Raster')

 plt.savefig(file2)

The plt.close() function works well to help me clean up images that have been previously run and saved.

Problem 4: Incorrect data output format

When writing the code to read the data, I ignored the type of data and instead the data was written as characters rather than integers when it was read, which resulted in a very confusing page alignment when I subsequently output the results file.

Solution:

I double-checked the code and added the separator split(',') to row\_data = line.strip().split(',') and then used row\_data = [int(x) for x in to format the read data as an integer row data. Writing this way ensures that the data is aligned and displayed in the output file. The modified code is as follows:

def read\_raster\_data(file\_path):

    with open(file\_path, "r") as f:

        lines = f.readlines()

    raster\_data = []

    for line in lines:

        row\_data = line.strip().split(',')

        row\_data = [int(x) for x in row\_data]

      raster\_data.append(row\_data)

return raster\_data

When reading data from a file, it is important to check that the input data is formatted as expected.

Problem 5: Unable to update the weighted results in real time

Since the initial weights are given i.e:

geology\_weight = tk.DoubleVar(value=0.5)

transport\_weight = tk.DoubleVar(value=0.3)

population\_weight = tk.DoubleVar(value=0.2)

However, due to an oversight on my part, I forgot to update the weighted data in a subsequent call, thus preventing the GUI interface from updating the weighted results in real time.

Solution:

After double-checking the code, a command was added to the code that creates the slider to call the update\_weights() function, which is used to update the weights of each factor. The changed function is as follows:

geology\_slider = tk.Scale(window, from\_=0.0, to=1.0, resolution=0.1, orient=tk.HORIZONTAL, label="Geology Weight", variable=geology\_weight, command=lambda event=None:update\_weights(), length=400,font=("Helvetica", 15))

Problem 6: The raster data is not flexible enough to read.

When I wrote the code to store the raster data, I fixed the n\_row and n\_cols of the raster data, which resulted in my program not being flexible enough to run if the user needed to replace it with other data to perform calculations. And writing it this way can easily result in cumbersome steps and redundant data, as it requires three visits to the same position of three raster data during the loop. As a result, it could lead to slow processing or program crashes if dealing with large data.

Solution:

Looking through the documentation, I found that I could read data from a file using the readlines() method, a function that allows me to work with raster datasets of different lengths. Use the split() method to split it into a list of values and convert the data into an integer format and store it as a two-dimensional list. With this rewrite, I think my program is now flexible enough in terms of reading and writing data to handle different types of raster data quickly.

The code is as follows:

def read\_raster\_data(file\_path):

    with open(file\_path, "r") as f:

        lines = f.readlines()  # Read all rows of data from a file

    raster\_data = []

    for line in lines:

        row\_data = line.strip().split(',')  # split into a set of numbers using the split(',') method

        row\_data = [int(x) for x in row\_data]  # Convert each numeric string to an intege

        raster\_data.append(row\_data) # Add each row of data as a one-dimensional list to the raster data list

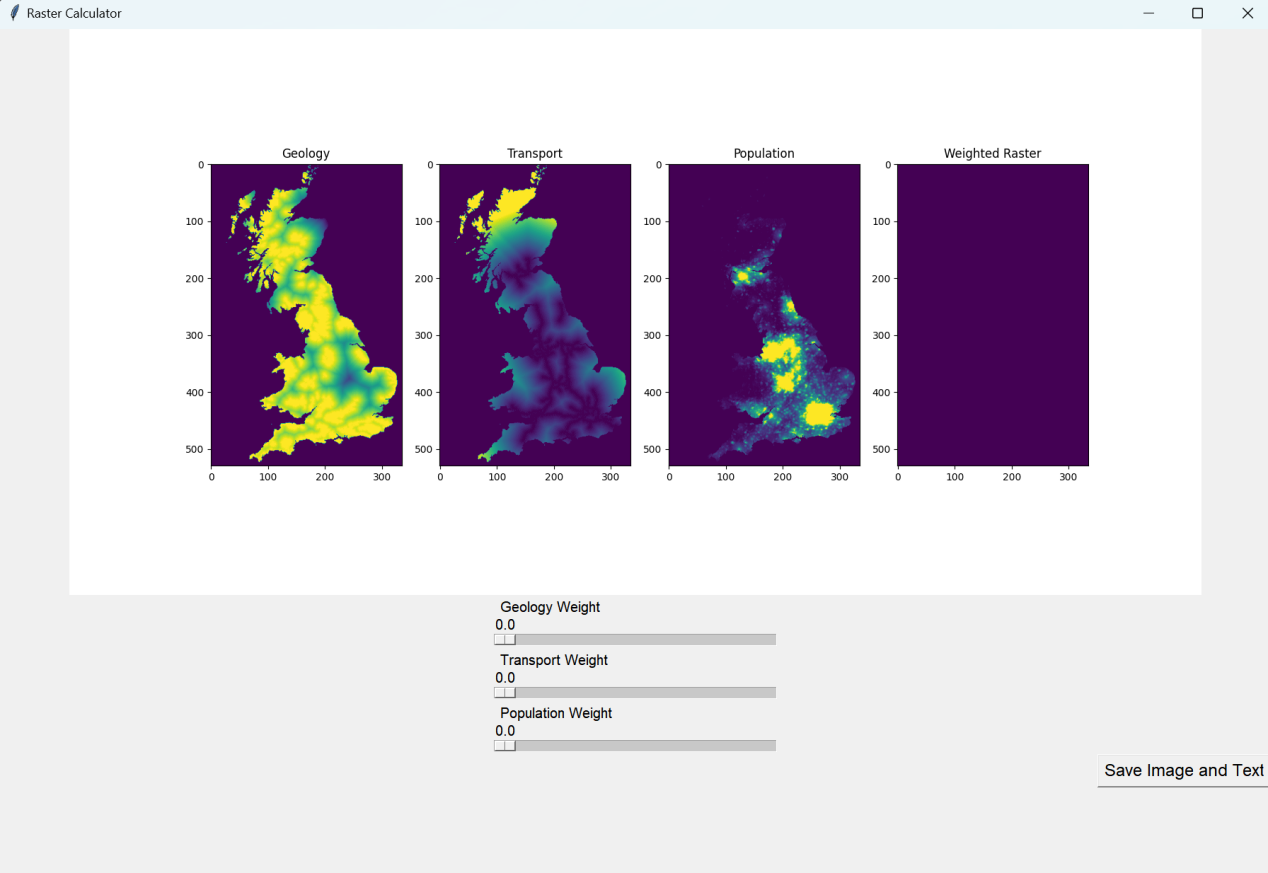
return raster\_data

Problem 7: The case where the sum of the sliders is 0.

When running the program, I found that if all three sliders were 0, the program would fail to calculate and an error would be reported.

Solution:

By reviewing the documentation, I found that I was writing the code with a missing judgement statement.So I rewrote the code to if total < 1 and total ! = 0, adding a condition to fix the runtime error.At this point, if the weighting factors are all zero, you will see an empty canvas, with no images on the graph. The GUI interface is shown in the figure.



Also I think I could add error handling to the code as sometimes there are cases where the file format does not match or the data is invalid. However, this idea has not yet been implemented by me, and I will continue to update my program as I go along.

###### Conclusion

Although most of the problems encountered when running the code have been successfully resolved by consulting the documentation, the program still lacks advanced features. There are many features waiting to be developed, such as adding widgets to allow users to zoom in on the results images and read the visualisations more clearly. There is also a need to optimise the software to enhance its performance to make it more advanced, to improve user-friendliness to incorporate more user interaction and to extend its compatibility across different platforms. For example, shortcuts to pages, such as shortcuts for quick saving, refreshing pages, etc., could be set up to make it easier for users to use the program. Overall, there is still a lot of room for improvement and the software development process will be a continuous and iterative one.