Assignment 2 document

The Black Death

GEOG5003 Programming for Geographical Information Analysis

The software has been designed and built to interactively model the number of weekly deaths in the Black Death outbreak in 1665. The weekly number of deaths is not known but can be modelled using the understood relationship between deaths and the number of rats caught per week in a 100-metre x 100-metre square and the average population density in a 100-metre x 100-metre square. Data on the average number of rats caught per week and average population density is available are utilised in this project to model the average weekly deaths in 1665. The model is incorporated into a piece of software to visualise the data and enable user interactivity.

The software performs four basic functions, sometimes repeatedly and with slight variations: read in the data, process the data, display the results and write the results to a text file.

**The data**

The software takes in files containing the average number of rats caught per week (rats.txt) and the average population density (parishes.txt). The population density averaged areas represent parishes, and the rats-caught averaged areas represent the domains of rat catchers. Each pixel represents a 100-metre x 100-metre area and the complete datasets represent a map of the study area. Using the csv package, the data are read using loops from text files and appended to nested lists within lists ([GEOG5003 I/O Practical](https://www.geog.leeds.ac.uk/courses/computing/practicals/python/agent-framework/part6/index.html)). The software displays a visualisation of the data on the screen. The matplotlib package is used to create and display the visualisation of the two maps. An appropriate colourmap is selected for the maps.

**Calculating average deaths**

The model iterates through the two datasets' values and passes them through an equation to calculate the average number of deaths per 100-metre x 100-metre square. Loops are utilised to iterate through the values within the nested lists for both lists. The equation calculating the number of deaths using this understood relationship is:

d = (0.8 x r) x (1.3 x p)

In this equation, *d* is the average number of deaths per week, *r* is the average number of rats caught per week, and *p* is the average population density. *r* and *p* are not equally weighted in this equation, so their values are multiplied by the weighting factor.

The loop carries out the equation and appends the result (*d*) to nested lists in a new list ([multiply nested list - stackoverflow.com](https://stackoverflow.com/questions/40888049/how-can-i-multiply-the-elements-of-two-nested-lists-in-python-where-the-sublists) and [append to nested lists – stackoverflow.com](https://stackoverflow.com/questions/58885772/how-to-multiply-nested-list-with-list)). The nested lists ensure the data still accurately represents the same geographic regions. The results are written to a new text file (average\_deaths.txt), with each line in the text file representing a line on the visualisation. The resulting dataset representing the calculated average number of deaths is then visualised as a map alongside the first two maps and displayed to the user.

**Interactive sliders**

Interactive elements are introduced into the software to allow modification by the user of the weighting factors in the average weekly deaths equation. Sliders are built using the IPython widgets library and allow the user to increase and decrease a value input ([instruction video](https://www.youtube.com/watch?v=SN0Bflf14C4)). By defining the weighting factors as variables in the script and inputting the variables into a slider each, the weighting factor can be adjusted by the user. The weighting factor variables are passed through the widgets using .**interactivity().**

*Issue*: The properties of the sliders are defined in the script, however, not all of these have applied properly to the slider. The default value sets correctly, but the min and max values do not seem to, nor does the description apply.

**Update function**

An update function (update\_plot) updates the deaths data each time the slider input value is modified ([How to update a plot in matplotlib – stackoverflow.com](https://stackoverflow.com/questions/4098131/how-to-update-a-plot-in-matplotlib)). Within this function, it reruns the deaths data calculation with the newly defined weighting factors. Each time the function is run, the deaths data is cleared first, otherwise the modified values would be added to the existing values instead of replacing them. The function then writes the results of the user-modified equation to a new text file (average\_deaths\_user\_modified.txt), which is then overwritten each time this input is modified. The update function writing new data to the text file appears to work as it should.

*Issue*: When the slider input values are modified and the calculated deaths values along with them, the update function should also update the visualisation on the screen, but this does not seem to work. I’ve attempted to troubleshoot the problem, and think I’ve narrowed it down to the incorrect use of matplotlib.**show**() in the function. I think that something like .**draw** should be used within the function and then .**show** used after/outside the function (when setting interactivity of the widgets) but I cannot seem to get it to work. ([How to update a plot in matplotlib – stackoverflow.com](https://stackoverflow.com/questions/4098131/how-to-update-a-plot-in-matplotlib))

**Overview**

Ideally, the software would be presented in a Graphical User Interface where all that is seen is perhaps the underlying data, methodology, the visualisations, and the interactive elements, but, importantly, the code is hidden from the user. User experience design is important to consider in software design to ensure the product is appropriate for the intended user. In this instance, it is probably unnecessary for the user to be able to see the model’s underlying code. Some additional statistics on the display may also be useful for the user; for instance actual values, rather than just a colour map of values.

Overall the piece of software works well modelling the average weekly deaths values with useful interactivity to modify the weighting factors in the equation. However, the major problem is that the final visualisation display does not update as the input data is updated.