**CSC2062 AIDA – Assignment 1**

**Section 1**

I used the code snippets provided in the practical sessions as well as a number of basic loops and conditions to read through every file in the folder where I store my PGM files. This information was then converted to a numpy array. Once this process had finished it was saved to a .csv file of the same name as the original file.

I was able to easily install and use GIMP to create my dataset and export it to PGM Files. I tried my best to keep the images fairly neat in order to prevent anomalous results, however, there remained a number of inconsistencies as I was using a computer mouse. I had struggled for the first number of days to successfully get Jupyter Notebook working on my home computer due to an issue with anaconda being unable to process file paths containing a space character. Once this had been overcome I was able to set it up relatively easily.

The coding of this section was straightforward, I was only hindered by my small logical errors such as mis-inputting the size of the 2D matrix causing my output .csv file to be too small. Many of these minor issues were caused by an unfamiliarity with python and lack of experience with file conversion. The greatest issue I experienced while coding this was the errors caused by not having the .pgm files in their own folder. This led to the creation of the try catch block I used to discover which I have left in the notebook in the case I needed to use it for further debugging.

**Section 2**

In my code I iterated through each file and used for loop nesting to get the value of each coordinate. I then used a number of if statements to apply the conditions specified in the assignment sheet. Once I had successfully stored all of the values I required into variables I assigned them to the respective row in the numpy array and once the program had successfully iterated through every file in the folder it saved it as 40403863\_features.csv.

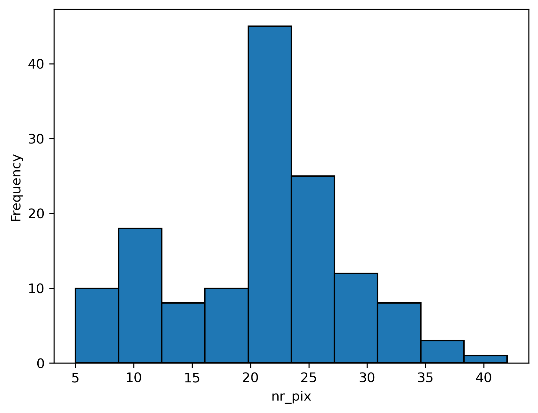
My custom feature was ‘densest\_quadrant’ which finds the midpoint of the drawn shape and goes through every pixel of the square and finds out where it is in relation to the midpoint. I then find the quadrant with the greatest amount of black pixels. Judging by the features csv this will prove very useful in differentiating between the letters.

Section 2 took me significantly longer to complete than section 1 due to substantially more difficult processes being required. I began by creating all of the necessary code such as numpy array and loading the first row with the feature names. I was now familiar with iterating through files having done it in the first section so this also spawned no issues. The logic required for automatically filling in the rest of the data was the cause of most of my problems in this section. There were a number of problems such as the neigh\_1 feature for which my for loop was too short and didn’t encompass all of the neighbours of the pixel, leading to almost of the values being 0. The fix for this caused all of the values to go to zero as the pixel was now counting itself as a neighbour. Other than this there were very few logical errors that I noticed and had to correct.

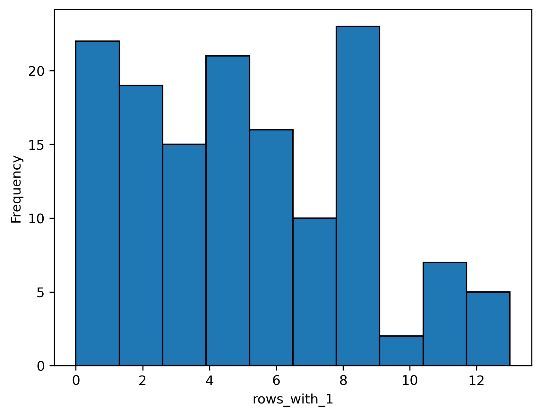
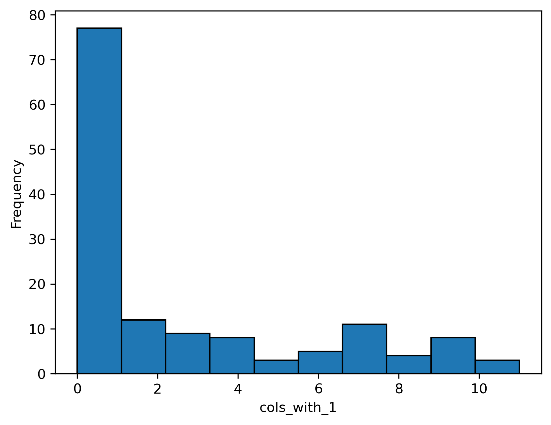
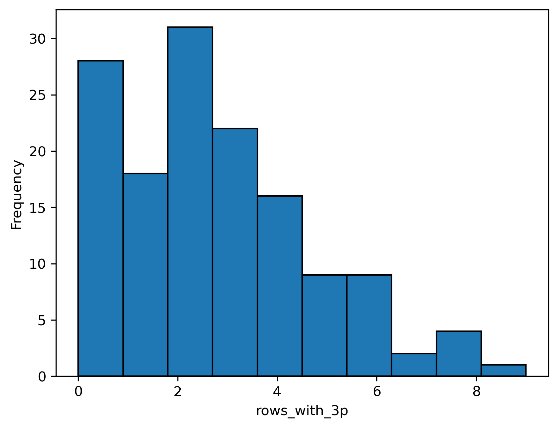
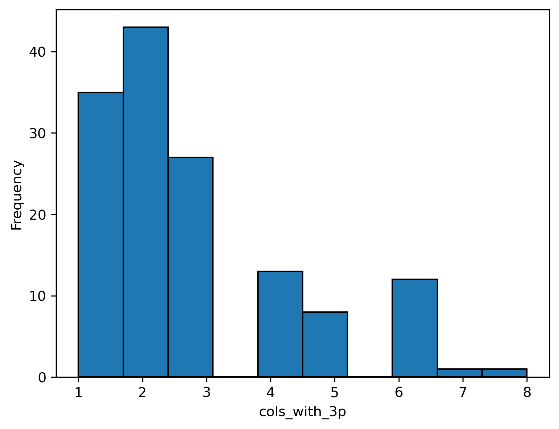
I found the connected\_areas and eyes features particularly because of the use of scipy. I spent a number of hours researching which libraries to use and how to use them. To learn this I used SciPy’s own documentation on the scipy.org website.

**Section 3**

**Section 3.1**

This part provided very few challenges as all of the necessary code was available through the practicals. The results are as follows.

Centre heavy with the dominant section being between 20 and 24 with over 40 of the 140 images having that amount. There is also a small crest towards the left which will be from the smaller characters such as i and xclaim.

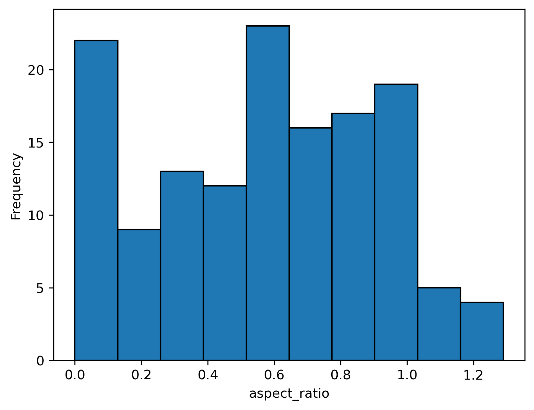


Left leaning graph suggesting a low amount of images with columns of 3 or more pixels. Interestingly this graph has no values less than one. Meaning that every image has at least one column with 3 or more pixels.

A relatively left leaning graph suggesting that most of the images have a low to moderate amount of rows with 3 or more pixels.

Very left heavy graph implying that very few images have columns with exactly one pixel which implies that most columns across the images have more than one pixel in them

Left leaning histogram with the peak in the middle.



A slightly right leaning even graph with its peak in the centre and a high crest at the far left caused my narrow characters such as i and xclaim. A vast majority of the values are less than 1 implying that most of the images are taller than they are wide. However, the right leaning nature suggests that they are often somewhat similar.