Project 1:

Submission deadlines: 5:00 pm, Friday 16th September, 2022
Value: **15%** of CITS1401.

To be completed individually.
You should construct a Python 3 program containing your solution to the following problem and submit your program electronically on Moodle. The name of the file containing your code should be your student ID e.g. 12345678.py. No other method of submission is allowed. Your program will be automatically run on Moodle for sample test cases provided in the project sheet if you click the "check" link. However, your submission will be tested thoroughly for grading purposes after the due date. Remember you need to submit the program as a single file and copy-paste the same program in the provided text box. You have only one attempt to submit so don't submit if you are not satisfied with your attempt. All open submissions at the time of the deadline will be automatically submitted. There is no way in the system to open the closed submission and reverse your submission.

You are expected to have read and understood the University's guidelines on academic conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort. Plagiarism detection, and other systems for detecting potential malpractice, will therefore be used. Besides, if what you submit is not your own work then you will have learnt little and will therefore, likely, fail the final exam.

You must submit your project before the deadline listed above. Following UWA policy, a late penalty of 5% will be deducted for each day (or part day), after the deadline, that the assignment is submitted. No submissions will be allowed after 7 days following the deadline except approved special consideration cases.

Overview

Face recognition (FR) is one the most widely known and used non-intrusive biometric which helps in identifying people. Given the identity of a known person, the goal of an FR system is to recognize the same person in a different scenario, for example while displaying a different expression. You can read this paper to understand more about FR under different expressions.

Your task is to help the researchers in analysing eight geodesic (surface) and eight 3D Euclidian distances between a few facial landmarks across four expressions ***'Neutral', 'Angry', 'Disgust', 'Happy'***. These distances on one face (in Neutral expression) can then be used to calculate similarity with the same face in different expressions or with other faces in the data set to see which faces are closer to (or look like) the reference face. Table 1 provides the details of each landmark while Figure 1 shows their location on the face. Table-2 gives you the details of the distances which will be used.

The attached csv (comma separated values) file has the eight geodesic and 3D Euclidian distances for the four expressions mentioned above for each person. For the sake of simplicity, the **Distance** numbers are given instead of their names. For example, *Inner-canthal width* is distance number 1 and Lower jaw width is distance number 8. Your task is to write a program which fulfills the following requirements.

***Table 1**: Details of the Facial landmarks.*

Landmark Name	Location
Ex	Outer eye corners
En	Inner eye corners
N	Nasal root
Ft	Temple points
Al	Outer edge of nostrils
Ch	Outer mouth corners
Go	Joint between the two jaws
Prn	Nose Tip
Ls	Midpoint of upper lip
Li	Midpoint of lower lip
Sto	Midpoint of both lips

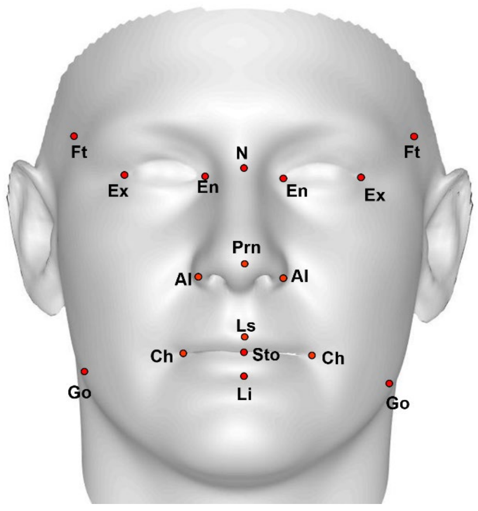

***Figure 1:** Facial landmarks locations on face*

Distance	Facial distance	LM1	LM2
1	Inner-canthal width	En_L	En_R
2	Outer-canthal width	Ex_L	Ex_R
3	Nasal bridge length	N	Prn
4	Nose width	Al_L	Al_R
5	Upper Lip thickness	Ls	Sto
6	Lower Lip thickness	Sto	Li
7	Forehead width	FT_L	FT_R
8	Lower jaw width	Go	Go

Table 2: List of distances used in this project. For example, Forehead width is the distance between Left Forehead (FT_L) and Right Forehead (FT_R). Similarly Inner canthal width, is the distance between the two inner eye corners: En_L and En_R.

Requirements (i.e. what your program should do)

Input:

Your program must define the function **main** with the following syntax:

```
def main(csvfile, adultID, Option):
```

The input arguments to this function are:

- `csvfile`: The name of the CSV file containing the data of individuals which needs to be analysed. Below are the first two rows of a sample file.

Adult ID	Expression	Distance	GDis	LDis
E001	Neutral	1	48.44795743	32.13047933

  The first row of the CSV file contains the headers "Adult ID", "Expression", "Distance", "GDis" and "LDis". From the second row, the first value of each row contains the de-identified adult ID, the expression displayed by the face, distance number (refer to Table-2), the geodesic distance and the 3D Euclidean distance. We do not have prior knowledge about the number of subjects we have to analyse (i.e. the number of rows) that the CSV file contains. Adult ID and Expression are strings, Distance is an integer while the remaining values are floats.
- `adultID`: The ID number of the adult we have to analyse. Remember that the ID is a string and is case sensitive.

- `Option`: The input argument that decides the type of analysis required. It can take only one of the two string inputs: "`stats`" or "`FR`". If the third input argument is "`stats`", then the objective of the program is to carry out some statistical analysis of the adult whose ID is given in the second argument. Otherwise, if the third input argument is "`FR`" then the objective is to perform face recognition by calculating the cosine similarity between the reference ID and other subjects.

Output:

The function is required to return the following outputs in the order provided below:

When the third input argument is "*`stats`*" then for the adult given in the input argument "adultID"

1. **OP1**- A list of lists containing the minimum (non-zero) and maximum GDis and LDis of each distance across the four expressions. For example, the minimum (non-zero) intercanthal width (geodesic and 3D Euclidean) in Neutral, Angry, Disgust and Happy expressions. There will be 8 lists inside the main list and each list will have four elements in the following order : [minimum GDis, maximum GDis, minimum LDis, maximum LDis]. The distances must be in the same order as given in Table-2.
2. **OP2**- A list of lists containing the difference between the geodesic and 3D Euclidean distances for each expression. There will be 4 lists inside the main list and each list will have eight elements.
3. **OP3**- A list containing the **average** geodesic distance of the eight distances across the four expressions. This list will have 8 elements.
4. **OP4**- A list containing the **standard deviation** of the 3D Euclidean distance of the eight distances across the four expressions. This list will have 8 elements. The formula to calculate standard deviation is provided at the end of this project sheet.

When the third input argument is "*`FR`*":

You will calculate the cosine similarity between the geodesic distances of the neutral expression of reference adultID and the geodesic distances of the remaining expressions of the same ID as well as the neutral expressions of the remaining subjects in the dataset. Your output will be:

1. The ID of the person with which the reference face has the maximum cosine similarity, and
2. The maximum cosine similarity value. The formula to calculate cosine similarity is provided at the end of this project sheet.

All returned lists should have the values in order. For example, for OP2, the main list has 4 sub-lists which should be ordered according to the expressions ***'Neutral', 'Angry', 'Disgust', 'Happy'.*** Inside each sub-list are 8 elements for the distances which must be ordered as per Table-2. All returned numeric outputs (both in lists and individual) must contain values rounded to four decimal places (if required to be rounded off). Do not round the values during calculations. Instead round them only at the time when you save them into the final output variables.

Examples:

Download the ***ExpData_Sample.csv*** file from the folder of Project 1 on LMS or Moodle. Some examples of how you can call your program from the Python shell (and examine the results it returns) are:

```
>>> OP1,OP2,OP3,OP4 = main('ExpData_Sample.csv','E001','stats')

>>> OP1
[[45.0087, 48.448, 29.9972, 32.9404], [104.1724, 110.913, 93.9636,
98.5776], [33.9933, 39.7572, 34.695, 40.0872], [49.5445, 66.477,
31.0483, 34.1107], [5.5607, 9.7036, 5.4345, 9.3448], [7.4147, 8.9331,
7.4147, 8.7645], [150.0258, 175.289, 118.1052, 127.0695], [137.8113,
154.4196, 107.0492, 111.6436]]

>>> OP2
[[16.3175, 10.2088, -0.5075, 34.6928, 0.3588, 0.0187, 48.2195, 46.5544],
[15.2932, 12.3354, -0.33, 18.4962, 0.1669, 0.1686, 39.2716, 45.8233],
[13.1529, 9.3969, -0.7016, 27.0089, 0.3791, 0.0, 39.5195, 36.9674],
[15.4137, 10.9684, -0.6347, 24.5052, 0.1262, 0.2101, 31.9206, 26.9959]]

>>> OP3
[46.7753, 106.8638, 37.9698, 58.3879, 7.5469, 8.1489,160.8882, 148.6113]

>>> OP4
[1.0775, 1.6565, 2.2229, 1.1442, 1.408, 0.6276, 3.5032, 1.8129]

>>> ID,cossim = main('ExpData_Sample.csv','E001','FR')

>>> ID
'A004'

>>> cossim
0.9992
```

Assumptions:

Your program can assume the following:

- Anything that is meant to be a string (i.e. header row) will be a string, and anything that is meant to be a number (i.e. data) will be a number.
- The order of columns in each row will follow the order of the headings provided in the first row.
- The data of each individual adult will always be contiguous. It will never be spread out randomly.
- The data for each expression will always be contiguous and in the order: ***'Neutral', 'Angry', 'Disgust', 'Happy'***.
- The data for distance may or may not be in order as in Table-2. For example, it is possible that the 6th distance of a particular expression is listed first and the 1st distance is listed last.
- Distances can never be negative or zero. If any distance is negative or equal to zero, then replace it with 50.
- No data will be missing in the csv file.
- The **main()** function will always be provided with valid input parameters.
- The formula for standard deviation and for calculating cosine similarity can be found at the end of the project sheet.

Important grading instruction:

Note that you have not been asked to write specific functions. This task has been left to you. However, it is essential that your program defines the top-level function *main(csvfile, adult, Option)* (hereafter referred to as "main()" to save space when writing it. Note that when "main()" is written it still implies that it is defined with its three input arguments). The idea is that within main(), the program calls the other functions. (Of course, these functions may then call further functions.) This is important because when your code is tested on Moodle, the testing program will call your main() function. So, if you fail to define main(), the testing program will not be able to test your code and your submission will be graded zero. Don't forget the submission guidelines provided at the start of the project sheet.

Things to avoid:

There are a few things for your program to avoid.

- **You are not allowed to import any Python module.** While use of many of these modules, e.g. csv or math is a perfectly sensible thing to do in production setting, it takes away much of the point of different aspects of the project, which is about getting practice opening text files, processing text file data, and use of basic Python structures, in this case lists and loops.
- Do not assume that the input file names will end in .csv. File name suffixes such as .csv and .txt are not mandatory in systems other than Microsoft Windows. Do not enforce that within your program that the file must end with a .csv or any other extension (or try to add an extension onto the provided csvfile argument), doing so can easily lead to loosing marks.
- Ensure your program does NOT call the *input()* function at any time. Calling the input() function will cause your program to hang, waiting for input that automated testing system will not provide (in fact, what will happen is that if the marking program detects the call(s), it will not test your code at all which may result in zero grade).
- Your program should also not call the *print()* function at any time. If it has encountered an error state and is exiting gracefully then your program needs to return zero as values for the cosine similarity otherwise empty lists or empty string. At no point should you print the program's outputs instead of (or in addition to) returning them or provide a printout of the program's progress in calculating such outputs.

Disclaimer: Although this project addresses a real-world problem, the files provided to you contain synthetically generated data.

Submission:

Submit your solution on Moodle before the deadline as per details provided at the start of the project sheet.

You need to contact unit coordinator if you have special considerations or you plan to be making a submission after the mentioned due date.

Marking Rubric:

Your program will be marked out of 30 (later scaled to be out of 15% of the final mark).

22 out of 30 marks will be awarded automatically based on how well your program completes a number of tests, reflecting normal use of the program, and also how the program handles various states including, but not limited to, different numbers of rows in the input file and / or any error states. You need to think creatively what your program may face. Your submission will be graded by data files other than the provided data file. Therefore, you need to be creative to look into corner or worst cases. I have provided few guidelines from ACS Accreditation manual at the end of the project sheet which will help you to understand the expectations.

8 out of 30 marks will be awarded on *style* (5/8) “the code is clear to read” and *efficiency* (3/8) “your program is well constructed and runs efficiently”. For style, think about use of comments, sensible variable names, your name and student ID at the top of the program, etc. (Please watch the lectures where this is discussed).

Style Rubric:

0	Gibberish, impossible to understand
1-2	Style is really poor or fair
3-4	Style is good or very good, with small lapses
5	Excellent style, really easy to read and follow

Your program will be traversing text files of various sizes (possibly including large csv files) so try to minimise the number of times your program looks at the same data items.

Efficiency Rubric:

0	Code too incomplete to judge efficiency, or wrong problem tackled
1	Very poor efficiency, additional loops, inappropriate use of readline()
2	Acceptable or good efficiency with some lapses
3	Excellent efficiency, should have no problem on large files, etc.

Automated Moodle testing is being used so that all submitted programs are being tested the same way. Sometimes it happens that there is one mistake in the program that means that no tests are passed. If the marker is able to spot the cause and fix it readily, then they are allowed to do that and your - now fixed - program will score whatever it scores from the tests, minus 4 marks, because other students will not have had the benefit of marker intervention. Still, that's way better than getting zero. On the other hand, if the bug is hard to fix, the marker needs to move on to other submissions..

Extract from Australian Computing Society Accreditation manual 2019:

As per Seoul Accord section D, a complex computing problem will normally have some or all of the following criteria:

- involves wide-ranging or conflicting technical, computing, and other issues;
- has no obvious solution, and requires conceptual thinking and innovative analysis to formulate suitable abstract models;
- a solution requires the use of in-depth computing or domain knowledge and an analytical approach that is based on well-founded principles;
- involves infrequently-encountered issues;
- is outside problems encompassed by standards and standard practice for

professional computing;
- involves diverse groups of stakeholders with widely varying needs;
- has significant consequences in a range of contexts;
- is a high-level problem possibly including many component parts or sub-problems;
- identification of a requirement or the cause of a problem is ill defined or unknown.

Formulas:

Standard deviation is mathematically expressed as:

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

σ = population standard deviation

N = the size of the population

x_i = each value from the population

μ = the population mean

You can find more details at https://en.wikipedia.org/wiki/Standard_deviation

Cosine Similarity Score

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Suppose we have two lists A and B that contains the geodesic between landmarks, A = {5, 2, 1,1, 3, 5, 2, 1} and B = {4,1,12, 9,7,4,1,12}

The cosine similarity of A and B can be calculated using the below formula and the answer will be:

$$similarity(A,B) = \frac{(5*4)+(2*1)+(1*12)+(1*9)+(3*7)+(5*4)+(2*1)+(1*12)}{\sqrt{5^2+2^2+1^2+1^2+3^2+5^2+2^2+1^2}*\sqrt{4^2+1^2+12^2+9^2+7^2+4^2+1+12^2}} = \frac{98}{\sqrt{70}*\sqrt{452}} = \frac{98}{177.8} =$$

0.5510

You can find more details at https://en.wikipedia.org/wiki/Cosine_similarity