CITS1401 Computational Thinking with Python

Project 2, Semester 1, 2025

Submission deadline: **Friday 23rd May 2025, 11:59 PM**

Total Marks: **30 (Value: 20%)**

Project description

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. **Please note that this is an individual project**. Your program will be automatically run on Moodle for some sample test cases provided in the project sheet if you click the "check" link. However, this does not test all required criteria, and your submission will be **manually** 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 do not submit until you are satisfied with your attempt. All open submissions at the time of the deadline will be automatically submitted. Once your attempt is submitted, there is no way in the system to open/reverse/modify it.

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 learned little and will therefore, likely, fail the final exam.

You must submit your project before the deadline mentioned above. Following UWA policy, a late penalty of 5% will be deducted for each day i.e., 24 hours after the deadline, that the assignment is submitted. No submissions will be allowed after 7 days following the deadline except approved special consideration cases.

Project Overview

Similar to Project 1, the data for this project is also the population information by areas and ages, distributed in two data files. You need to find the correct data association across files. The files include the codes and names of Australian states, statistical areas (Level 2 & 3), and different age population groups living in these areas. The map and relationship between statistical areas Level 2 and 3 is presented in Figure 1, and details can be found at https://www.abs.gov.au/statistics/standards/australian-statistical-geography-standard-asgs-edition-3/jul2021-jun2026/main-structure-and-greater-capital-city-statistical-areas/statistical-area-level-2

Task: You are required to write a Python 3 program that will read two CSV files. After reading the files, your program is required to analyse the data. More details are given in the Output specification section.

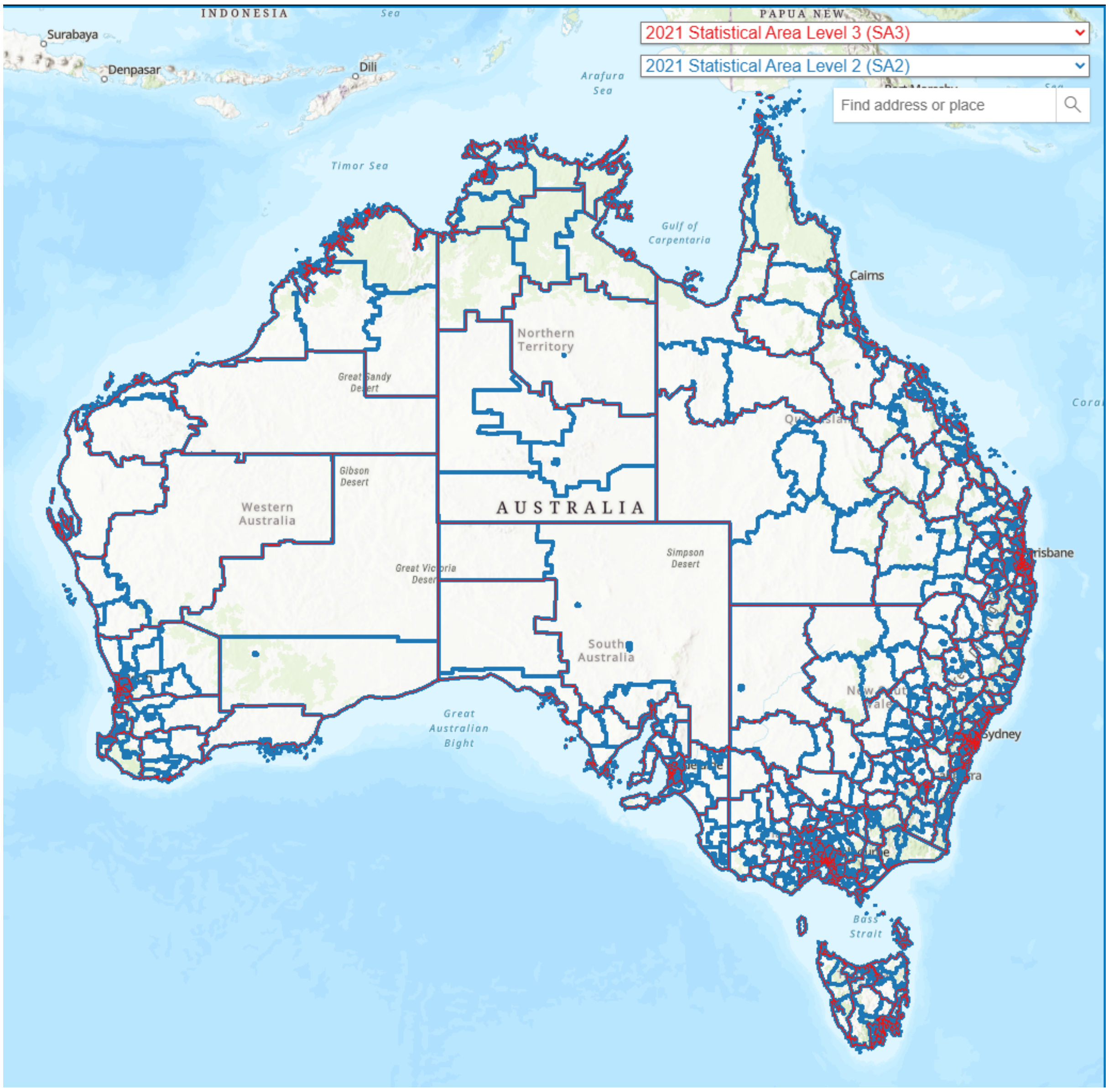

Figure 1: Australian map showing the boundaries of different statistical areas as mentioned in the legend.

Requirements

1) **You are not allowed to import any external or internal module in python**. While use of many of these modules, e.g., `csv` or `math` is a perfectly sensible thing to do in a 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, e.g., loops, lists, and dictionaries.
2) 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 the 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).

3) Your program should also not call `print()` function at any time except for the case of graceful termination (if needed). If your program has encountered an error state and is exiting gracefully then your program needs to return `zero` (for number), `None` (for string), empty list (for list), or empty dictionary (for dictionary) and print an appropriate message. 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.
4) 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 csv file argument), doing so can easily lead to syntax error and losing marks.

Input

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

```
def main(csvfile_1, csvfile_2):
```

The input arguments for this function are:

- **IP1** `(csvfile_1)`: The name of the CSV file (as string) containing the relationship data between different statistical levels of areas for each state. The first row of the CSV file will contain the headings of the columns. A sample CSV file "`SampleData_Areas_P2.csv`" is provided with the project sheet on LMS and Moodle.
- **IP2** `(csvfile_2)`: The name of the CSV file (as string) containing the record of the population. The first row of the CSV file will contain the headings of the columns. A sample CSV file "`SampleData_Populations_P2.csv`" is provided with the project sheet on LMS and Moodle.

Output

We expect 3 outputs in the order below.

OP1: A dictionary, where keys are all the age groups found in the data, as formatted strings of format '`lower-upper`', e.g., '`0-9`' and '`80-None`'. Use '`None`' if one of the bounds does not exist. The value for each key of the dictionary needs to be a list including following three items in the order below:

1. The name of the state with the largest population in the age group (key), across all the states in the data.
2. The name of the SA3 area with the largest population in the age group (key), across all the SA3 areas in the data.

3. The name of the SA2 area with the largest population in the age group (key), across all the SA2 areas in the data.

When there are multiple states/areas with the same largest population, choose the first one in alphabetical order in terms of state/area code. The codes of states/areas should be treated as strings.

OP2: A nested dictionary where keys of the outer dictionary are the codes of all the states present in the data, and the value for each key will be a dictionary. For the inner dictionaries (which are values of outer dictionary), the keys will be the codes of all SA3 areas of the state (outer key) with populations of 150000 or higher in the state (outer dictionary's key). The value of the inner dictionary will be a list containing following three elements in the order below:

1. the code of the SA2 area with the largest population, across all the SA2 areas within the SA3 areas identified by the key of the inner dictionary.
2. the total population of this SA2 area [found in (OP2:1)].
3. the standard deviation of populations across age groups in this SA2 area [found in (OP2:1)].

When there are multiple areas with the same largest population, choose the first one in alphabetical order in terms of area code. The codes of states/areas should be treated as strings. Use empty inner dictionary if there is no SA3 area with populations equal or greater than 150000.

OP3: A dictionary where the keys are the names of all the SA3 areas present in the data, that have 15 or more SA2 areas. The value will be a list, indicating a pair of SA2 areas within the SA3 area, with the maximum similarity (measured by Cosine Similarity) between the percentages of populations in each age group. The list includes following items in the order below:

1. the name of first SA2 area in the pair (the name ranks first alphabetically).
2. the name of second SA2 area in the pair (the name ranks second alphabetically).
3. the Cosine Similarity score (a float number) between the percentages of populations in each age group of the two SA2 areas.

Use empty dictionary if there is no SA3 area with 15 or more SA2 areas.

All returned numeric outputs 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 `SampleData_Areas_P2.csv` file and `SampleData_Populations_P2.csv` file from the folder of Project 2 on LMS or Moodle. An example of how you can call your program from the Python shell and examine the results it returns, is provided below:

```
>> OP1,OP2,OP3=main('SampleData_Areas_P2.csv','SampleData_Populations_P2.csv')
```

The returned output variables are:

```
>> OP1
{
'0-9': ['western australia', 'wanneroo', 'baldivis - south'],
'10-19': ['western australia', 'wanneroo', 'baldivis - south'],
'20-29': ['western australia', 'stirling', 'adelaide'],
'30-39': ['western australia', 'stirling', 'enfield - blair athol'],
'40-49': ['western australia', 'wanneroo', 'baldivis - south'],
'50-59': ['western australia', 'wanneroo', 'mindarie - quinns rocks - jindalee'],
'60-69': ['western australia', 'onkaparinga', 'glenelg (sa)'],
'70-79': ['western australia', 'onkaparinga', 'victor harbor'],
'80-None': ['western australia', 'stirling', 'victor harbor']
}
>> OP2
{
'4': {'40304': ['403041072', 16431, 483.8303]},
'5':  {'50503':  ['505031099',  25259,  1002.5155],  '50403':  ['504031057',  24301,
1305.5413], '50502': ['505021089', 24126, 677.144], '50501': ['505011072', 16908,
602.2749]},
'6': {}
}
>> OP3
{
'wanneroo': ['carramar', 'landsdale', 0.9975],
'onkaparinga': ['aldinga', 'seaford rise - moana', 0.9992],
'swan': ['beechboro', 'lockridge - kiara', 0.9974],
'stirling': ['dianella - south', 'yokine - coolbinia - menora', 0.9984],
'launceston': ['legana', 'riverside', 0.9982]
}
```

Assumptions

Your program can assume the following:

- The order of columns can be different from the order provided in the sample file. Also there can be extra columns in the CSV file.
- Age groups will be non-overlapping. They can vary and be different from the sample file.
- Rows can be in random order except the first row containing the headings in both CSV files. The header names can be in lower, upper, or mixed cases.

- All string data in the CSV files is case-insensitive, which means "Perth" is same as "perth". Your program needs to handle the situation to consider both to be the same. All string outputs are also treated case-insensitive.
- There can be missing or invalid data, for example, missing values, negative populations, invalid formats, duplicated rows, etc. You need to think of other cases yourself. If there is any invalid data, then entire row(s) should be ignored. For duplicated rows, all the rows should be ignored. If there are invalid/duplicated rows in one of the two CSV files, the corresponding rows in both files should be ignored. Handling missing or invalid data should happen before any calculation of the outputs.
- Values can be zero and must be accounted for mathematical calculations. Your program must not crash.
- Number of states, SA3 areas, SA2 areas, and age groups will vary, so do not hardcode.
- The `main` function may not be provided with valid input parameters.
- The necessary formulas are provided at the end of this document.

Debugging Documentation and Reflection

As a crucial part of developing computational thinking and programming skills, you are required to document your debugging process while coding. This will help you gain insights into programming errors and strengthen your problem-solving abilities. All programs go through debugging process, and we would like to see your learning about debugging process.

You should provide detailed documentation for each significant issue, including:

- **Error Description**: Copy the error message for syntax errors, or describe the unexpected behaviour for semantic errors.
- **Erroneous Code Snippet**: Copy the line(s) of code where the issue was identified.
- **Test Case**: Provide the specific test case (inputs) that triggered the error, and values of variables in the erroneous code snippet if relevant.
- **Reflection (Maximum 75 Words):** Reflect on why the error occurred, what is your reasoning process to solve the error, and/or what you learned from the debugging process.

Your debugging documentation must be included at the end of your submitted Python script (both the text box and the file) as multiline comments (`""" Comment here """`).

Provide documentation for **three** significant debugging issues encountered during development. Your debugging documentation and reflection will be manually reviewed and assessed based on relevance and clarity.

Example:

```
"""
Debugging Documentation:
Issue 1 (Date 2025 May 20):
```

```
- Error Description:

      TypeError: unsupported operand type(s) for +: 'int' and 'str'

- Erroneous Code Snippet:

      population_sum = population_sum + population_list[i]  # Line 85

- Test Case:

      main('SampleData_Areas.csv', 'SampleData_Populations.csv')    # The
      Inputs

      # You can also give more details, such as the variable values in the
      erroneous code snippet, for example:

            population_sum: 1452

            population_list[i]: '672'

- Reflection:

      I realized that population data read from file was stored as strings.
      To fix the bug, I added integer conversion using int() before
      calculation. Learned that I need to be careful about data types before
      performing arithmetic operations.
"""
```

Important grading instruction

Note that you have not been asked to write specific functions. The task has been left to you. However, it is essential that your program defines the top-level function `main(csvfile_1, csvfile_2)` (hereafter referred to as "`main()`" in the project document to save space when writing it. Note that when `main()` is written, it still implies that it is defined with its 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 this document.

Marking rubric

Your program will be marked out of 30 (later scaled to be out of 20% of the final mark). 20 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 error or corner 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 investigate 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.

10 out of 30 marks will be awarded on debugging process (6/10), style (2/10) "the code is clear to read" and efficiency (2/10) "your program is well constructed and run efficiently". For style, think about use of proper comments, function docstrings, sensible variable names, your name and student ID at the top of the program, etc. (Please watch the lectures where this is discussed).

Debugging Process Rubric:

0	The issue is unclear or irrelevant to the submitted code.
1	The issue is clearly documented and relevant to the submitted code, but the reflection is vague.
2	The issue is clearly documented and relevant to the submitted code, and the reflection is meaningful.

Each of the three debugging issues in your documentation will be assessed individually based on the above rubric.

Style Rubric:

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

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

Efficiency rubric:

0	Code too complicated to judge efficiency or wrong problem tackled.
1	Very poor efficiency, additional loops, inappropriate use of `readline()`,etc.
2	Good efficiency and works well with large files, etc.

Automated 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 can 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 per intervention, 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.

Necessary formulas

i. **Cosine Similarity:**

Mathematical formula to calculate Cosine Similarity is as follows:

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\cdot\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

where $\mathbf{A}$ and $\mathbf{B}$ are two data vectors, and A_i and B_i are individual dimensions. More details can be found on https://en.wikipedia.org/wiki/Cosine_similarity

ii. **Standard deviation, *s*:**

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}}$$

where x_1, x_2, x_3 … … x_n are observed values in the sample data. $\bar{x}$ is the average value of observations and N is the number of observations.