**Project 1, Semester 1, 2025**

Submission deadline: **Friday 18th April 2025, 11:59 PM**

Total Marks: **30 (Value: 12%)**

## 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

**Background:** The Australian Bureau of Statistics (ABS) has commissioned your services to develop a powerful data analysis tool that can provide insights into the population distribution across Australia's cities and regions. The government and urban planners require this tool to make informed decisions regarding infrastructure development, resource allocation, and future urban planning. Your task is to develop an analytical tool that processes real-world population datasets and generates insightful statistics.

**Data**: The data for this project is 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

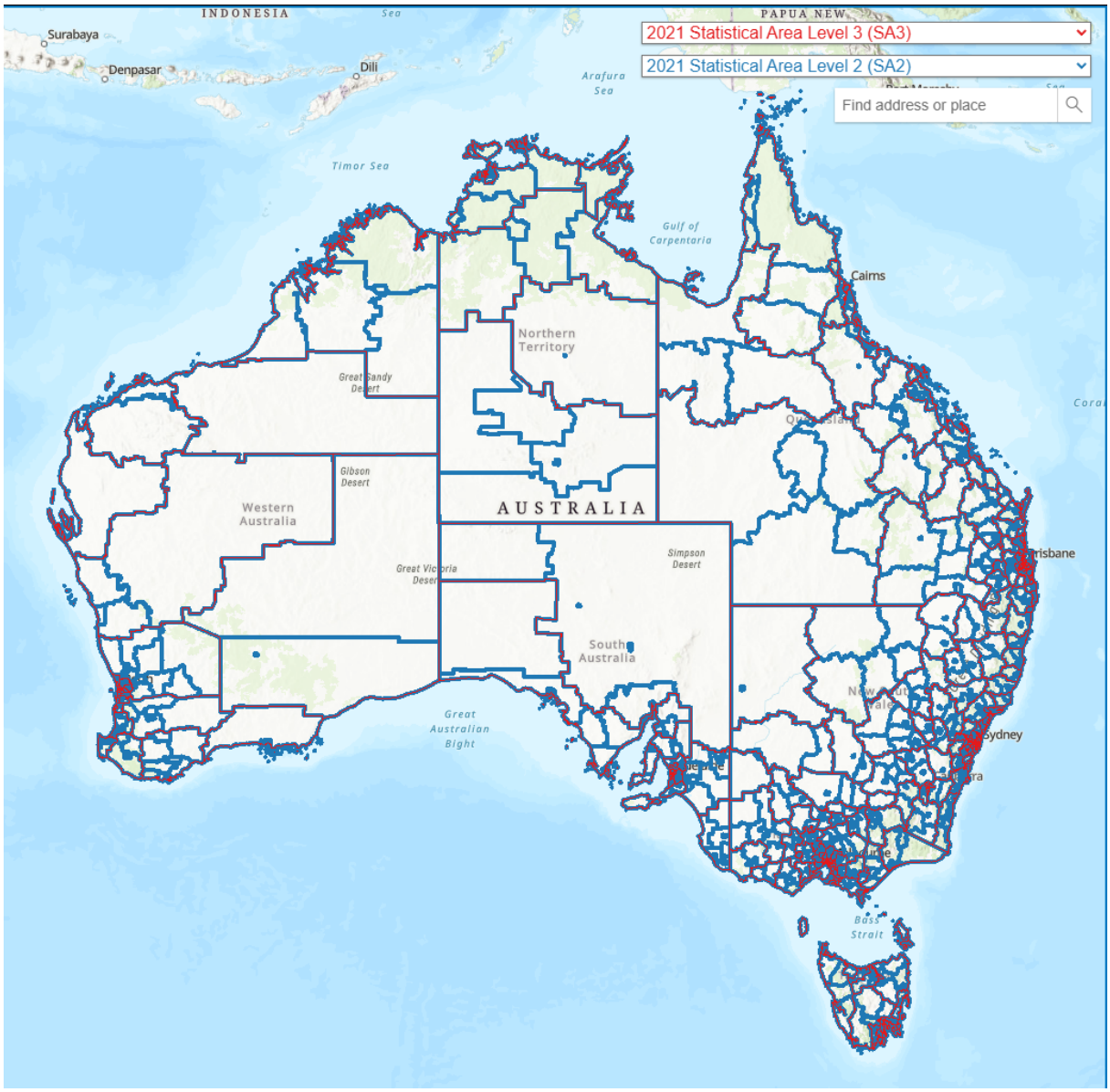


Figure.7¿Australian.map.showing.the.boundaries.of.different.statistical.areas.as.mentioned.in.the.legend¡

**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 complete the following tasks. More details are given in the Output specification section.

1) Find the age group that contains a specific input age.
2) Calculate population statistics for two specific SA3 areas.
3) Find the SA3 area with the largest population in the age group, for each unique state, and its percentage.
4) Calculate the correlation between the age structure of two specific SA2 areas.

## 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, in this case lists and loops.
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), or empty list (for list) 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, age, sa2_1, sa2_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.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.csv"` is provided with the project sheet on LMS and Moodle.
- **IP3** `(age)`: Age as an integer.
- **IP4** `(sa2_1)`: String containing the code of an SA2 area.
- **IP5** `(sa2_2)`: String containing the code of another SA2 area.

## Output

We expect 4 outputs in the order below.

**OP1:** Output will be a list including two integers, indicating the lower bound and the upper bound of the *age group* containing the input age (**IP3:** `age`). Use `None` as the list element if one of the bounds does not exist.

Note: The *age group* identified in this output will be used to calculate the other outputs.

**OP2:** Output will be a list of two lists.

The first list includes three elements in the order below:

1. the code of the SA3 area where input **IP4** (`sa2_1`) is located,
2. the average of populations in the *identified age group* (**OP1**), across all SA2 areas in this SA3 area,
3. the standard deviations of populations in the *identified age group* (**OP1**), across all SA2 areas in this SA3 area.

The second list is similar to the first one, but for input **IP5** (sa2_2).

**OP3:** Output will be a list of list(s). Each inner list corresponds to a unique state in the data, including three elements in the order below:

1. the state name,
2. the name of the SA3 area with the largest population in the *identified age group* (**OP1**), in the state,
3. the percentage of the population which you found above with respect to the total population across all age groups in the same SA3 area.

The inner list(s) should be sorted in alphabetically ascending order by the state name. Hint: Look for *sort()* or *sorted()* function.

When there are multiple areas with the same largest population, choose the first one in alphabetical order in terms of area code.

**OP4:** Output will be a float number which is the correlation coefficient between the populations in each age group in the first input **IP4** (`sa2_1`), and the second input **IP5** (`sa2_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 `SampleData_Areas.csv` and `SampleData_Populations.csv` files from the folder of Project 1 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, OP4 = main('SampleData_Areas.csv',
'SampleData_Populations.csv', 18, '401011001', '401021003')
```

The returned output variables are:

```
>> OP1

[15, 19]

>> OP2

[['40101', 782.5, 376.8879], ['40102', 689.625, 493.9609]]

>> OP3

[['south australia', 'onkaparinga', 0.0595], ['tasmania', 'launceston',
0.0591], ['western australia', 'wanneroo', 0.0694]]

>> OP4

0.0154
```

## Assumptions

Your program can assume the following:

- Anything that is meant to be string (e.g., names and codes of states and areas) will be a string, and anything that is meant to be numeric will be numeric in the CSV file.
- 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. Similarly, your program needs to handle the string input parameter in the same way. All string outputs are also treated case-insensitive. The output must contain all strings in lower case.
- The order of columns in each row will follow the order of the headings provided in the first row. However, rows can be in random order except the first row containing the headings.
- No data will be missing in the CSV files; however, values can be zero and must be accounted for mathematical calculations.
  [In case any part of the calculation cannot be performed due to zero values or other boundary conditions, do a graceful termination by printing an error message and

returning a zero value (for number), `None` (for string) or empty list (for list) depending on the data type. **Your program must not crash.**]
- Number of states, SA3 areas, SA2 areas, will vary, so do not hardcode.
- The `main` function will always 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.

**Sample:**

```
"""
Debugging Documentation:
Issue 1 (Date 2025 April 12):
- 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',     18,
    '401011001', '401021003')  # 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, age, sa2_1, sa2_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 four 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 12% of the final mark). 20 out of 30 marks will be awarded 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 or corner states/cases. 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. Few guidelines from ACS Accreditation manual are provided 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 understand" 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 or very basic 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 highly 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 | 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 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. **Correlation coefficient, *r*:**

Mathematical formula to calculate correlation is as follows:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

where $x_i$ and $y_i$ are the individual sample points. $\bar{x}$ and $\bar{y}$ are the sample means.

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.