CITS1401 Computational Thinking with Python
Project 2 Semester 2 2021

Project 2:

Submission deadlines: 5:00 pm, Friday 22nd October 2021
Value: **20%** of the total marks of CITS1401.

To be done 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. Following 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 submission deadline listed above. Following UWA policy, a late penalty of 5% will be deducted for each day (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.

Overview

As we know from earlier project that social media is one of the most attractive marketing platform nowadays. Therefore, the advertisements need to be more relevant and interesting to the audience. Many different factors are considered to adapt the advertisements for each and every user.

In this project, you are requested to write a computer program that can read the data from a CSV (comma-separated values) file provided to you and return different interesting analytical results. Your program should follow the following specification.

Specification

You need to follow the below instructions of the project input and output.

INPUT

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

```
def main(inputFile,[locId1,locId2],radius):
```

The input arguments are:

- `inputFile` is the name of the CSV file containing the information and record about the location points which need to be analysed for this project. The first row of the CSV file will contain the many attributes but we are interested in the following four attributes:
  - `LocId`: A unique ID of a location point.
  - `Latitude`: Latitude of the location point.
  - `Longitude`: Longitude of the location point.
  - `Category`: Location Types which can be many e.g., Parking (P), Hospital (H), Restaurant (R), Chemist Shop (C), Super Market (S), etc.
- `[locID1,locID2]` is an input parameter that accepts a list of two strings which represent two location IDs.
- `radius` is the numeric input parameter that defines a circular boundary around the location IDs provided in `queryLocId`.

OUTPUT

Based on the inputs mentioned above, firstly you need to define two circular regions of radius `radius` with center at `locID1` and `locID2` which we label as C1 and C2 respectively. Figure 1 presents the scenario for understanding.

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

- Return a list containing two dictionaries (let's call them D1 and D2), where the keys in the dictionaries are the location categories, and their values contain the number of locations for respective category in region C1 and C2 respectively.
- Return the cosine similarity of the regions C1 and C2 based on the category-wise number of locations identified inside each region. The formula to calculate cosine similarity is provided at the end of this project sheet.
- Return a dictionary of category based on the common location IDs existing in the regions C1 and C2.
- Return a list of two dictionaries: one for each input location ID (i.e., `locID1` and `locID1`), where the key of each dictionary item will be location category and the value will be a tuple containing the Location ID of the

respective category which is closest to the respective input location ID and its distance from the same input location ID. The formulate to calculate distance is provided at the end of this project sheet.

All returned output variables containing numerical values should be rounded to four decimal places (if required to be rounded off). Remember not to round the values during calculations and round them only at the time of saving them in the output variables

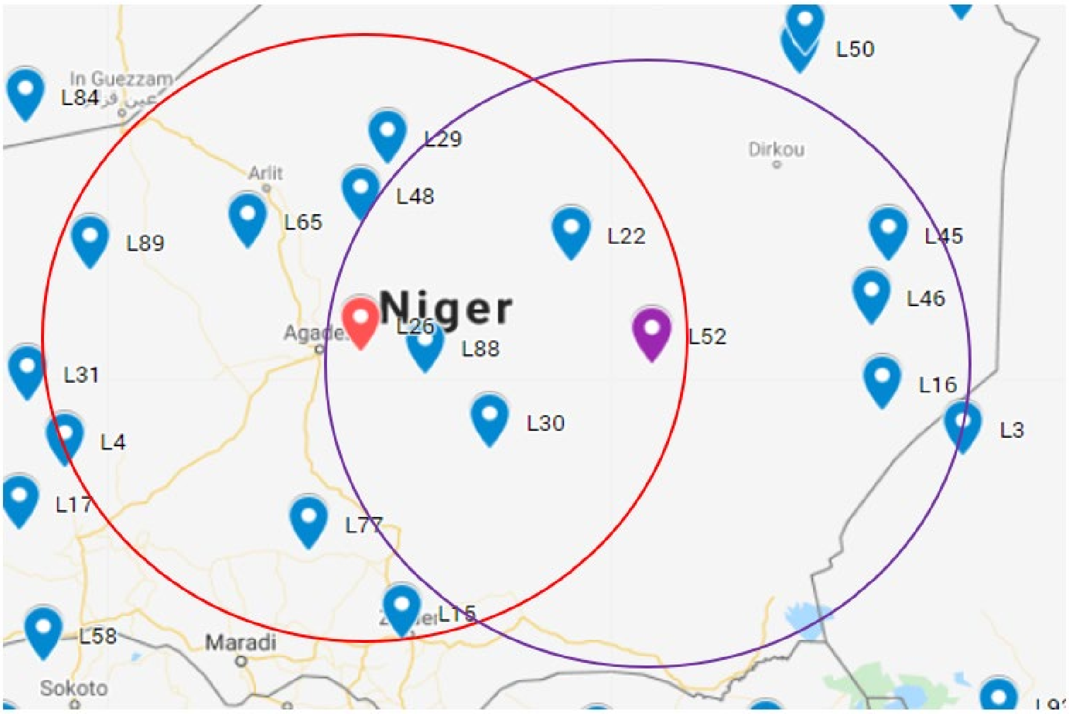

Figure 1: Two defined circular regions C1 and C2 in red and violet centered at location L26 and L52 respectively.

Example

Download the *Locations.csv* file from the folder of Project 1 on LMS or Moodle. An Example of the interactions are given below:

```
>>> LDCount1, simScore1, DCommon1, LDClose1 = main("Locations.csv",
["L26", "L52"], 3.5)
```

The output variables of the program will be as below:

```
>>> LDCount1 = [{'P': 1, 'H': 3, 'R': 2, 'C': 2, 'S': 3}, {'P':
3, 'H': 2, 'R': 1, 'C': 0, 'S': 2}]

>>> simScore1
0.7711

>>> DCommon1 = {'P': ['L26'], 'H': ['L52', 'L22'], 'R': ['L88'],
'C': [], 'S': ['L30']}
```

```
>>> LDClose1 = [{'H': ('L77', 2.3034), 'R': ('L88', 0.7736), 'C':
('L29', 2.0607), 'S': ('L65', 1.556)}, {'P': ('L46', 2.4717),
'H': ('L22', 1.4374), 'R': ('L88', 2.5338), 'S': ('L30',
2.0482)}]
```

Another example of interaction is given below:

```
>>> LDCount2, simScore2, DCommon2, LDClose2 = main ("Locations.csv" ,
["L89", "L15"], 4.3)
```

The output variables of the program will be as below:

```
>>> LDCount2
[{'P': 3, 'H': 2, 'R': 3, 'C': 4, 'S': 1}, {'P': 2, 'H': 4, 'R':
3, 'C': 1, 'S': 4}]

>>> simScore2
0.7319

>>> DCommon2
{'P': ['L26'], 'H': ['L77'], 'R': ['L88'], 'C': ['L4'], 'S': []}

>>> LDClose2
[{'P': ('L31', 1.6058), 'H': ('L17', 3.0011), 'R': ('L48',
3.0555), 'C': ('L84', 1.7586), 'S': ('L65', 2.1059)}, {'P':
('L26', 3.2431), 'H': ('L77', 1.4124), 'R': ('L38', 2.5739), 'C':
('L4', 4.2148), 'S': ('L30', 2.3415)}]
```

Additional requirements:

There are few more requirements for your program.

- Your program needs to validate the inputs to the main() function and gracefully terminate if invalid inputs are provided.
- You program needs to terminate *gracefully* if the file cannot be found or opened. For graceful terminations, you need to print the message related to the problem and return `None` for each unavailable output.
- Your program needs to validate the input data from the file. If data is not correct then entire row needs to be discarded.
- Your program needs to consider that columns and rows of the csv file do not have any specific order or can be in any order (excluding header row).
- Your program needs to interpret the header row to find the required columns. Your program needs to terminate *gracefully* if required columns cannot be found in the file.
- Your program needs to convert all text data in the csv to be lower order alphabets to ensure that program is case insensitive.
- Your program needs to check that location ID is unique otherwise row should be considered as corrupt data.

Important grading instruction:

You will have noticed that you have not been asked to write specific functions. That has been left to you. However, it is important that your program must defines the top-level function `main()`. The idea is that within `main()`, the program calls the other functions. (Of course, these may call further functions.) The reason this is important is that when your program is tested, 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 program 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 the 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.
- 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.
- Ensure your program does NOT call the `input()` or `print()` functions at any time (`print()` function can be used for graceful terminations only). That will cause your program to hang, waiting for input that automated testing system will not provide. In fact, what will happen is that the marking program detects the call(s), and will not test your code at all which may result in zero grade.

Submission:

Submit your solution before the deadline electronically on Moodle. 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 purpose by teaching team after the due date. Remember you need to submit the program as a single Python file and copy-paste the same program in the provided text box. You have only one attempt to make the submission so don't submit if you are not satisfied with your attempt. You are encouraged to keep your attempt open as there is no way in the system to open the closed submission and reverse your submission. All open submissions at the time of deadline will be automatically submitted except special consideration or late submissions.

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

Marking Rubric:

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

22 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 error states, different number of rows in the input file or missing data for months/days, etc. You need to think creatively what your program may face. Your submission will be graded by data files other than the provided sample 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". Remember, the expected standards are much higher than those of earlier project.

For style, think about use of comments, sensible variable names, your name and student ID at the top of the program, etc. (Please watch 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 you need 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 data reading or use of `readline()`
2	Acceptable or good efficiency with some lapses
3	Excellent efficiency, should have no problem on 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 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.

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 distionaries A and B that contains the category-wise number of locations, A = {“P”: 5, “H”:2, “R”:1, “C”:0, S: “3”} and B = {“P”: 4, “H”:0, “R”:12, “C”:9, “S”:7}

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

$$similarity(A,B) = \frac{(5*4)+(2*0)+(1*12)+(0*9)+(3*7)}{\sqrt{5^2+2^2+1^2+0^2+3^2}*\sqrt{4^2+0^2+12^2+9^2+7^2}} = \frac{53}{\sqrt{39}*\sqrt{290}} = \frac{53}{106.3485} = 0.4984$$

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

Euclidean Distrance

The Distance (*d*) between two points A = (x_1, y_1) and B = (x_2, y_2) in cartesian plain is calculated as,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

In this project, a location point is considered as A = (longitude, latitude).

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