Department of Computer Science and Software Engineering

CITS1401 Computational Thinking with Python

Project 1: Computing World Happiness Index

Submission deadline: **5:00pm Friday 20 September 2019** for the code, and **5:00pm Friday 06 September 2019** for the pseudocode
Value: 15% 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 LMS. No other method of submission is allowed.

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 submission deadline listed above. Following UWA policy, a late penalty of 10% will be deducted for each 24 hours (day), after the deadline, that the assignment is submitted. However, in order to facilitate marking of the assignments in a timely manner, no submissions will be allowed after 7 days following the deadline. Remember, you need to inform Unit Coordinator if you are submitting your project after the deadline to ensure that it is graded.

Overview

For the last few years, the United Nations Sustainable Development Solutions Network has been publishing the World Happiness Report. Details of the 2018 report can be found here. The underlying data, which you can also download from the latter URL, is a combination of data from specially commissioned surveys undertaken by the Gallup organisation, and statistical and economic data from other sources. The web site linked above also provides the methodology for how the different data have been combined to compute the final score, most dramatically called the Life Ladder.

Here is a sample:

country	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Confidence in national government
Afghanistan	2.66171813	7.460143566	0.490880072	52.33952713	0.427010864	-0.106340349	0.261178523
Albania	4.639548302	9.373718262	0.637698293	69.05165863	0.74961102	-0.035140377	0.457737535
Algeria	5.248912334	9.540244102	0.806753874	65.69918823	0.436670482	-0.194670126	
Argentina	6.039330006	9.843519211	0.906699121	67.53870392	0.831966162	-0.186299905	0.305430293
Armenia	4.287736416	9.034710884	0.697924912	65.12568665	0.613697052	-0.132166177	0.246900991
Australia	7.25703764	10.71182728	0.949957848	72.78334045	0.910550177	0.301693261	0.45340696

The data shown above (and discussed below) can be found in the CSV formated text file [WHR2018Chapter2_reduced_sample.csv]

The actual method used to compute the Life Ladder score is quite complicated, so the the aim of this Project, in brief, is to test whether simpler methods can yield similar results. In particular, the Project aims to see whether any of a range of proposed methods yields a similar ranking, when countries are ranked by Life Ladder score in descending order i.e. from happiest on these measures, to least happy. (The Wikipedia article also discusses criticisms of the World Happiness Report process.)

Looking at the data sample above, you can see that the column headers occupy the first row, the countries are listed in the first column, while the Life Ladder scores that we are seeking to emulate are in the second column. The third and subsequent columns contain the data from which you will compute your own Life Ladder scores. However, for this exercise, please remember that the aim is not to replicate the precise Life Ladder scores, but rather to replicate the ranking of countries as a result of the Life Ladder scores.

Eye-balling the Data

In Data Science projects, it is always a good idea to "eyeball" the data before you attempt to analyse it. The aim is to spot any trends ("this looks interesting") or any issues. So, looking at the sample above (ignoring the first two columns), what do you notice?

There is a difference in scale across the columns. Healthy Life Expectancy at Birth ranges from 52.3 to 72.8, but in general is valued in 10's, while Social Support is a value in the range 0.0 to 1.0, and Freedom to Make Life Choices has both negative and positive floating point numbers. (The problem of GDP per Capita being actually valued in the thousands, or tens of thousands, has already been solved by the data collectors taking logs.) The issue is that you don't want a particular attribute to appear significant just because it has much larger values than other attributes.
The other thing you may have noticed is that sometimes the data is simply missing, e.g. the score for Confidence in National Government for Algeria. Any metric we propose will have to deal with such missing data (which is actually a very common problem).
Specification: What your program will need to do

Input
Your program needs to call the Python function `input` three times to:

get the name of the input data file
get the name of the metric to be computed across the normalised data for each country. The allowed names are "min", "mean", "median" and "harmonic_mean".
get the name of the action to be performed. The two options here are: "list", list the countries in descending order of the computed metric, or "correlation", use Spearman's rank correlation coefficient to compute the correlation between ranks according to the computed metric and the ranks according to the Life Ladder score.

The order of the 3 calls is clearly important.

Output
The output, printed to standard output, will be either a listing of the countries in descending order based on the computed metric, or a statement containing the correlation value (a number between -1.0 and 1.0).

Tasks: A more detailed specification

Use `input` to read in 3 strings, representing the input file name, the metric to be applied to the data from the file (**excluding the first two columns**) and the action to be taken to report to the user.
Read in the CSV formated text file. That is, fields in each row are separated by commas, e.g.

```
Albania,4.639548302,9.373718262,0.637698293,69.05165863,0.74961102,-0.035140377,0.457737535
Algeria,5.248912334,9.540244102,0.806753874,65.69918823,0.436670482,-0.194670126,
```

Apart from the first field, all the other fields are either numbers (so converted using `float()`, or empty, which can be translated to the Python object `None`. Each line will be transformed into a row, represented as a list, so you end up with a list of lists.

For each column apart from the first two, compute the largest and smallest values in the column (ignoring any `None` values).
Given the maximum and minimum values for each column, normalise all the values in the respective columns. That is, each value should be normalised by transforming it to a value between 0.0 and 1.0, where 0.0 corresponds to the smallest value, and 1.0 to the largest, with other values falling somewhere between 0.0 and 1.0. For example, the minimum Life Expectancy years in the small dataset is 52.33952713. This is transformed to 0.0. The maximum value is 72.78334045, which is transformed to 1.0. So, working proportionally, 69.05165863 is transformed to 0.81746645. In general, the transformation is *(score - min)/(max-min)*, where max and min are the respective maximum and minimum scores for a given column, and will, of course, differ from column to column.
For each row, across all the columns except the first two, compute the nominated metric using the normalised values (excluding `None`). "min", "mean" and "median" are, respectively, the minimum value (on the basis that a nation's happiness is bounded by the thing the citizens are grumpiest about), "mean" and "median" are the arithmetic mean and median value (discussed here). The harmonic mean of a list of numbers is defined here. For harmonic mean, apart from avoiding `None` values, you will also have to avoid any zeroes; the other metrics have no problem with 0. The output from this stage is a list of country,score pairs.
The list of country,score pairs are either to be listed in order of descending score, or the Spearman's rank correlation coefficient should be computed between the country,score list that you have computed and the Life Ladder list, when sorted by descending score. You can assume there are no tied ranks, which means that the simpler form of the Spearman calculation can be used. An example of how to compute Spearman's rank correlation can be found here.

Example

```
>>> happiness.main()
Enter name of file containing World Happiness computation data:
WHR2018Chapter2_reduced_sample.csv
Choose metric to be tested from: min, mean, median, harmonic_mean mean
Chose action to be performed on the data using the specified metric. Options are
list, correlation correlation
The correlation coefficient between the study ranking and the ranking using the mean
metric is 0.8286

>>> happiness.main()
Enter name of file containing World Happiness computation data:
WHR2018Chapter2_reduced_sample.csv
Choose metric to be tested from: min, mean, median, harmonic_mean harmonic_mean
Chose action to be performed on the data using the specified metric. Options are
list, correlation list
Ranked list of countries' happiness scores based the harmonic_mean metric
Australia 0.9965
Albania 0.5146
Armenia 0.3046
Afghanistan 0.0981
Argentina 0.0884
Algeria 0.0733
```

The complete table is in file WHR2018Chapter2_reduced.csv

Important

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 defines the top-level function** `main()`. The idea is that within `main()` the program calls the other functions, as described above. (Of course, these may call further functions.) The reason this is important is that when I test your program, my testing program will call your `main()` function. So, if you fail to define `main()`, my program will not be able to test your program.

Assumptions

Your program can assume a number of things:

Anything is that meant to be a string (i.e. a name) will be a string, and anything that is meant to be a number (i.e. a score for a country) will be a number.
The order of columns in each row will follow the order of the headings, though data in particular columns may be missing in some rows.

What being said, there are number of error conditions that your program should explicitly test for and respond to. One example is detecting whether the named input file exists; for example, the user may have mistyped the name. The way this test can be done is to first:

```
import os
```

Then, assuming the file name is in variable `input_filename`, use the test:

```
if not os.path.isfile(input_filename) :
   return(None)
```

and test for `None` in the calling function (likely `main()`).

Things to avoid

There are a couple things for your program to avoid.

Please do **not** import **any** Python module, other than `os`. While use of the many of these modules, e.g. `csv` or `scipy` 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.
Please 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.
Please make sure your program has **only 3** calls to the `input()` function. More than 3 will cause your program to hang, waiting for input that my automated testing system will not provide. In fact, what will happen is that the marking program detects the multiple calls, and will not test your code at all.

Submission

Stage 1: Submit a single PDF file containing your approach and pseudocode for the solution of the problem as per guidelines discussed in Lecture L2 Software Development Process. You need to discuss the document with lab demonstrator before submission. It is mandatory to submit this file before **5:00pm 06 September 2019** on LMS to avoid 10% deduction in Project 1 grading. This will be a formative feedback of your problem solving skills developed in the course. In case you do not submit the file, 10% of the total marks of the project will be deducted from your obtained grade of the Stage 2 submission.

Stage 2: Submit a single Python (`.py`) file containing all of your functions via LMS.

You need to contact unit coordinator if you have special considerations or making late submission.

Marking Rubric

For convenience, your program will be marked out of 30 (later scaled to be out of 15% of the final mark).

22 out of 30 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 error states, such as the input file not being present. Other than things that you were asked to assume, you need to think creatively about the inputs your program may face.
8 out of 30 will be ***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 at the top of the program. (Please look at your lecture notes, where this is discussed.)

Style Rubric

0	Gibberish, impossible to understand
1-2	Style is really poor, or your have imported 2 or more extraneous modules
3-4	Style is good or very good, with small lapses, or your have imported an extraneous module
5	Excellent style, really easy to read and follow

For Project 1, there are not too many ways your code can be inefficient, but try to minimise the number of times your program looks at the same data items. There are particular places where you should use `readline()`, but not in a loop.

Efficiency Rubric

0	Code too incomplete to judge efficiency, or wrong problem tackled
1	Very poor efficiency, addtional loops, inappropriate use of readline()
2	Acceptable efficiency, one or more lapses
3	Good efficiency, within the scope of the assignment and where the class is up to

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 2 marks, because other students will not have had the benefit of marker intervention. Still, that's way better than getting zero, right? (On the other hand, if the bug is too hard to fix, the marker needs to move on to other submissions.)

10% (3 marks) will be deducted if the stage 1 submission is not made.