CITS1401 Computational Thinking with Python
Project 2 Semester 1 2020

Project 1:

Submission deadlines: 5:00pm, Friday 29th May 2020
Value: **20%** 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 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

Benford's law is a mathematical rule on how frequent certain digits appear in numbers. For example, it is significantly more likely to see a 1 as the first digit in a number than a 9. For the second digit in a number, 1 is still more likely than 9, but the difference is less. In general, the further down the digit, the more balanced the distribution should be. As this law describes what sort of distributions of digits, we are likely to see, it can be used in detecting fraudulence.

Forensic accounting is about trying to detect the difference between normally occurring numbers, and numbers that have been changed in order to falsify an account. Due to the natural randomness in such accounts, it cannot be determined which individual numbers are fraudulent and which are legitimate. However, the digits over many numbers can be examined to build up a large distribution. If this distribution looks significantly different to the one predicted by Benford's Law, there is some evidence that the account is fraudulent.

The task for this project is to write a program that can calculate the distribution of digits for numbers found in a certain file. You are not asked to actually detect any fraudulence.

Specification: What your program will need to do

Input:

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

```
def main(filename,no_places,regularise=False):
```

- `filename`: This input argument is the name of the csv (comma separated values) file (a text file in which each value is separated by a comma ',' character - and each 'row' of values is separated by a new line) containing data which includes numerical data which needs to be analysed. There is also no prior knowledge about the number of rows, the number of elements in each row or the number of elements which will be "numerical" in the csv file. Your code will need to analyse the entire file but ensure it analyses only numerical values which include integers and decimal numbers only.
- `no_places`: This input argument provides the information about the number of places in each numerical data to analyse. For example, if this value is 1, we are only examining the distributions of the first digits (from left hand side) in each numerical data. If the value is 3, we are examining the first, second, and third digits (from left hand side) of each numerical data. Each digit place will have its own distribution.
- `regularise`: This input argument is an optional parameter with a default value of `False`. When set to `True`, the distribution values should represent the distribution as a fraction of the total sum for each digit place in the output instead of the count. See below for an example of how your program should work with this parameter set to True.

Output:

The function is required to return the list of lists. Each sub-list (inner list) represents the distribution of digits for that digit place. Each sub-list should be of length 10, and represents the counts of each of the digits 1, 2, 3, 4, 5, 6, 7, 8, 9, and 0 respectively (in that order).

All returned lists must contain numerical values rounded to four decimal places (if required). Remember not to round the values during calculations and round them only at the time of saving them in the output list.

For example, if the number of places input (`no_places`) is 3, then it is expected that the output to be a list of 3 sub-lists. Each sub-list represents the distribution of digits in the first, second, and third digit places respectively. Each sub-list would have a length of 10.

Example:

Download the sample_accounts.csv file from the folder of Project 2 on LMS. A sample interaction is:

Sample 1:

```
>>> output = main("sample_accounts.csv",1)
```

The output returned in the variable is:

```
>>> output
[[26, 22, 28, 22, 16, 20, 31, 22, 13, 0]]
```

Sample 2:

```
>>> output = main("sample_accounts.csv",1,True)
```

The output returned in the variable is:

```
>>> output
[[0.13, 0.11, 0.14, 0.11, 0.08, 0.1, 0.155, 0.11, 0.065, 0.0]]
```

Sample 3:

```
>>> output = main("sample_accounts.csv",3)
```

The output returned in the variable is:

```
>>> output
[[26, 22, 28, 22, 16, 20, 31, 22, 13, 0], [17, 21, 23, 15, 15, 23,
30, 18, 14, 23], [14, 12, 23, 22, 22, 8, 21, 24, 25, 28]]
```

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. You may provide an error message followed by an empty list as an output.
- Your program needs to validate the input data from the file and it should analyse the integer part of the data only. Note float numbers digit counts should consist of only the digits to the left of the decimal point (e.g. the number "`52.385686`" is considered to have 2 digits and therefore only '`5`' and '`2`' should be counted by your program).
- Your program needs to consider that all numeric data in the file will not be of same length so distribution will, obviously, not count higher digits (when reading digit numbers from left to right) on the numeric data that does not contain these digits (e.g. if `no_places` = 5 and your program is processing the number "`268`" then it will not count anything for digit places 4 and 5). For instance, if you sum all the elements of each sub-list of Sample 3 output (presented above), you will observe that sum of first sub-list is greater than sum of second and third sub-lists. This is due to the reason that there is a numerical data in the csv file which is of single digit only.

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 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 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 and your submission may be graded zero.

Things to avoid:

There are a couple 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 numpy 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.
- 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()`function at any time. That 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 call(s), and will not test your code at all which may result in zero grade.
- Ensure your program does NOT call the `print()` function except in the scenarios where you need to gracefully terminate the program. No outputs should be printed.

Submission:

Submit a single Python (.py) file containing all of your functions via LMS before **5:00 pm 29 May 2020** on LMS. The submission system is available in the folder *Project 2* on LMS.

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

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 (such as different number of columns and rows in the input file) as well as different error states. You need to think creatively what your program may face.

8 out of 30 marks 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, etc. (Please look at your lecture notes, 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. You may think to use different data structure such as tuples, lists, or dictionaries.

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 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 too hard to fix, the marker needs to move on to other submissions.