CS2204 Homework: Fraudulent Transactions

Objectives

- Learn to read text files with custom formatting
- Learn how to sort tuples with selected field(s)
- Learn to use the Python standard library documentation
- Implement simple statistical and filtering algorithms

Background

In this assignment, you are going to implement and combine simple algorithms for finding potentially fraudulent credit card activities. You are given a text database of financial transactions (`transactions.txt`), where each line contains a single transaction. An example line from the file:

```
2019-08-01 23:54:09 | $47.10 | Miller Ltd | +1-109-904-2332
```

The four fields of the transaction are separated by a `|` character and some optional white spaces. The first field is the date and time of the transaction (using the *24-hour* format), the second field is the USD amount, the third is the name of the merchant, and the last field is the merchant's phone number. It is a good idea to open this file in Spyder to better understand the format. Also, you can re-use some parts of your previous homework for processing this file.

You are going to read this database and implement several filtering functions to find irregular activities. Finally, you will combine these methods to find potentially fraudulent transactions.

Tasks

You need to finish the `fraud.py` source file. Note: you can use the simple tests provided at the end of the file (by uncommenting those lines). Feel free to change/add/delete any testing code. The outputs of this testing code are not going to be graded. The validator program will use your code directly.

You are already given a new data type, called `Transaction` , with the following fields:

1. `time` : should be a **string** value, containing the date and time of the transaction
2. `amount` : should be a **float** value, for the USD amount
3. `company` : should be a **string** value, the name of the merchant
4. `phone` : should be a **string** value, the phone number of the merchant with a _+country*code* prefix.

1. Implement the `load_transactions` function. Instead of storing the results in a global variable, return a list of `Transaction` instances created by parsing the database. Make sure you do the proper type conversion of the field(s) where needed. (15 pts)

2. Implement the `foreign_transactions` function. As a simplification, foreign transactions are those where the merchant's phone number does not start with the `+1` country code. Check the documentation string (and the example testing code) for the expected behavior. (10 pts)
3. Implement the `late_night_transactions` function. Check the documentation string (and the example testing code) for the expected behavior. Also, you can find some additional hints below. (15 pts)
4. Implement the `highest_transactions` function. You need to return the top n transaction objects, based on the USD amounts. Check the documentation string (and the example testing code) for the expected behavior. (15 pts)
5. Implement the `median_expense` function. You need to return a single float, which is the median value (https://en.wikipedia.org/wiki/Median) of the USD amounts in the provided transactions list. (15 pts).
6. Implement the `significant_transactions` function. You need to return those transactions where the USD amount is greater than or equal to **five times** of the **median** spending for a trailing number of (previous) transactions. The number of trailing transactions is provided as a parameter. Obviously, you cannot detect a *significant transaction* for the first trailing number of transactions (not enough data to calculate the median) (15 pts).
7. Implement the `fraudulent_transactions` function to combine the results of the previous algorithms. Check the documentation string (and the example testing code) for the expected behavior. (15 pts)

Hints

As always, you can use any approach and/or Python libraries installed with the standard Anaconda distribution. For working with date and time information, I highly recommend to check out the documentation of the `datetime` (https://docs.python.org/3/library/datetime.html) module available in the standard library. More specifically, the similarly named `datetime` (https://docs.python.org/3/library/datetime.html#datetime.datetime) datatype inside this module and its strptime() (https://docs.python.org/3/library/datetime.html#datetime.datetime.strptime) method can be helpful. This is a good opportunity to learn to read Python documentation (as a practicing Python programmer, you will do this a lot!).

Grading

You can use the attached `validator.py` program to check your work (and the instructor's original mistakes). It will also estimate your final score for the homework. The program is in the same folder as your homework assignment. Open the `validator.py` script and run it in the Spyder environment to track your progress.

Penalties

Points will be deducted if you fail to set `__author__` variable (-10 pts) and for **each PEP 8 style errors** (-1 pt for each) in your program.

Submission

Please, upload the final version of the following file(s) (**and only those files**) to Brightspace:

- `fraud.py`