## Petal Data Engineering Problem Set

Write a script to process the *transactions[1-3].csv.gz* files.

These files each contain 100k bank and credit card transactions for ~100 users (not real Petal users). The files are sorted by user id, but not by anything else. Each row has 7 columns and is properly formatted (although you will need to infer the exact csv formatting params yourself).

1. user id
2. account id (this column can be ignored)
3. debit/credit (debit means negative amount. credit means positive amount)
4. amount (this field should always be positive)
5. Random string with tricky characters to parse (where a transaction description would normally go)
6. Date
7. Another random string with tricky characters to parse

**The desired output is a csv with the columns:**

1. user\_id
2. Number of of transactions for user
3. The sum of transaction amounts for the user (use exactly 2 decimal places). This is the same as the final balance.
4. The min balance (running sum) for the user at the end of any day (use exactly 2 decimal places). This value should be at most $0 since it starts there.
5. The max balance (running sum) for the user at the end of any day (use exactly 2 decimal places). This value should be at least $0 since it starts there.

**Example:**

*10/15 credit $5*

*10/16 credit $3*

*10/17 nothing happens*

*10/18 debit $6*

*10/19 credit $10*

*10/19 debit $2*

*10/20 debit $1*

For this user, the output should be:

number of transactions: 6

sum of transactions: 9

minimum balance: 0

maximum balance: 10 (Yes, it’s 10, not 12 because there are 2 transactions on 10/19)

**WARNING! Please make sure you understand the above example. Specifically, why maximum balance is 10.**

The output should only include one row per user and with users in the original input order.

Use the header: “user\_id,n,sum,min,max” and don’t use any extra whitespace so we can easily run a diff with our reference solution.

Even though these example files are small, you should pretend they have too much data to fit in memory. But you can assume that the transactions for a single user will fit in memory. So your script should do one of the following:

1. Stream the data (you can use the fact that the files come grouped by user id) so that you don’t run out of memory.
2. Create a barebones in-memory MapReduce framework with a memory efficient mapper and reducer. The framework piece can be as inefficient as you want :)

**Parallelism (Bonus):**

1. If you chose streaming then add coarse grained parallelism so that multiple files can be processed at once. But a single file is processed in a single threaded way. You should have one output file per input file.
2. If you chose MapReduce then make sure you can run as many mappers as there are input files and a bunch of reducers. You should have one output file per reducer (don’t worry about the output order of the users if you choose MapReduce).

**Common Pitfalls:**

1. Don’t worry about handling users on the edges of files since they can span from the end of one file into the beginning of the next file.
2. The files are sorted by user id, but not by anything else.
3. Make sure that all balances are output using exactly 2 decimal places.
4. If you use Python: Parallelism in Python is kind of weird. Make sure you’re actually getting good CPU utilization.