Affirm Coding Challenge: Balance the Loan Books

Background

- At Affirm, we borrow money from our banking partners through **debt facilities**. In turn, we use these facilities to extend **loans** to customers.
- A banking partner may require a **covenant**, which is a set of restrictions on the loans that a facility may fund. Common restrictions include:
    - Maximum default rate: A bank may restrict us from funding certain riskier loans.
    - Geographic location: A bank may restrict us from funding loans in certain states.
- We may establish multiple facilities with a single bank. In this case, the bank may define covenants that apply to all of their facilities, or only to an individual one.

Goals

You will be provided with a list of facilities and covenants, as well as a stream of loans that Affirm would like to fund with those facilities. Your task is to write a program that consumes loans from the stream and assigns each loan to a facility while respecting each facility's covenants.

Input

An input data set will consist of four CSV files, describing the facilities, banks, covenants, and loans, respectively. These files are described in the following sections. You will be given two data sets, a small data set (in the folder 'small') for manually verifying your understanding of the problem, along with a large data set (in the folder 'large') for more rigorously stress-testing your program. The folder 'small' will also contain the solution files 'assignments.csv' and 'yields.csv'. These files will be described later.

facilities.csv

Each row in this file describes a single facility.

Field	Type	Description
`bank_id`	integer	The ID of the bank providing this facility.
`facility_id`	integer	The ID of the facility.
`interest_rate`	float	Between 0 and 1; the interest rate of this facility. In this simplified model, when we

		use `x` dollars from this facility to fund a loan, we are charged `x * interest_rate` dollars in interest.
`amount`	integer	The total capacity of the facility in cents.

banks.csv

Each row in this file describes a banking partner.

Field	Type	Description
`bank_id`	integer	The ID of the bank.
`bank_name`	string	The name of the bank.

covenants.csv

Each row in this denormalized file represents at least one covenant that we have with a bank. If a row contains both a `max_default_likelihood` and a `banned_state`, they should be treated as separate covenants.

Field	Type	Description
`bank_id`	integer	The ID of the bank requiring this covenant.
`facility_id`	integer	If present, denotes that this covenant applies to the facility with this ID; otherwise, this covenant applies to all of the bank's facilities.
`max_default_likelihood`	float	If present, specifies the maximum allowed default rate for loans in the facility (or in the bank's facilities).

`banned_state`	string	If present, indicates that loans in the facility (or in the bank's facilities) may not originate from this state.

loans.csv

Each row in this file represents a loan we would like to fund. Loans are ordered chronologically based on when we received them.

Field	Type	Description
`id`	integer	The ID of the loan. Strictly increasing.
`amount`	integer	The size of the loan in cents.
`interest_rate`	float	Between 0 and 1; the interest rate of the loan. In this simplified model, the amount of money we earn from a loan (if it doesn't default) is `amount` * `interest_rate`.
`default_likelihood`	float	Between 0 and 1; the probability that this loan will default. In this simplified model, when the loan defaults, we lose all the money that we lent and do not earn any interest on the loan.
`state`	string	State where the loan originated.

Calculating Loan Yields

The *expected yield* of a loan funded by a facility is the amount of interest that we expect to earn from the loan (taking into account the chance of a default), minus the expected loss from a default, minus the interest that we pay to to the bank to use their facility:

```
expected_yield =
    (1 - default_likelihood) * loan_interest_rate * amount
    - default_likelihood * amount
    - facility_interest_rate * amount
```

Guarantees

Our FP&A team has been hard at work trying to negotiate good facility deals with our banking partners. They have guaranteed us that the following constraints hold:

1. We can fund all the loans by processing them in the order that they are received and assigning each loan to the cheapest facility that can accommodate that loan legally (i.e. satisfying all the required covenants and not exceeding the facility's capacity). *Note: depending on your ordering of facilities with equal interest rate, you may be unable to assign the last loan (#425) of the large data set. This is OK.*
2. The expected yield of funding any loan with any facility is always nonnegative.

Deliverables

Your program should consume the input data and attempt to fund each loan with a facility. Unfunded loans are ignored by our system -- they will earn no interest, nor will they lose money if they default. Your program should be *streaming*, meaning it that it should process loans in the order that they are received and not use future loans to determine how the current loan should be funded.

Please include instructions on how to run (and build, if necessary) your code.
Your program should produce two output files:

assignments.csv

Each row in this file describes a loan assignment.

Field	Type	Description
`loan_id`	integer	The ID of the loan.
`facility_id`	integer	If the loan is funded, the ID of its facility; otherwise, empty.

yields.csv

Each row in this file describes the expected yield of a facility.

Field	Type	Description
`facility_id`	integer	The ID of the facility.

`expected_yield`	int	The expected yield of the facility, rounded to the nearest cent. This is defined as the sum of the expected yields for all the loans in the facility.

After completing a working code solution, you should include a write-up answering the following questions to the best of your ability:

1. How long did you spend working on the problem? What did you find to be the most difficult part?
2. How would you modify your data model or code to account for an eventual introduction of new, as-of-yet unknown types of covenants, beyond just maximum default likelihood and state restrictions?
3. How would you architect your solution as a production service wherein new facilities can be introduced at arbitrary points in time. Assume these facilities become available by the finance team emailing your team and describing the addition with a new set of CSVs.
4. Your solution most likely simulates the streaming process by directly calling a method in your code to process the loans inside of a for loop. What would a REST API look like for this same service? Stakeholders using the API will need, at a minimum, to be able to request a loan be assigned to a facility, and read the funding status of a loan, as well as query the capacities remaining in facilities.
5. How might you improve your assignment algorithm if you were permitted to assign loans in batch rather than streaming? We are not looking for code here, but pseudo code or description of a revised algorithm appreciated.
6. Discuss your solution's runtime complexity.

Feel free to include any additional comments in your write-up.

You should send us a folder containing the source code to your program, the output files produced by your program on the 'large' data set, as well as your analysis.

Evaluation

Your program will be evaluated based on the following criteria, in decreasing order of importance:

1. *Correctness* – Does the assignment produced by your program satisfy all the required covenants? Does the assignment stay within each facility's capacity?
2. *Clarity* – Is your code well-organized and easy to read?

3. *Extensibility* – Is your solution architected in a manner that makes it easy to collaborate with multiple engineers, and allow them to add and modify features in a consistent, testable way?

Final Notes and Suggestions

Important! Please read carefully:

- *We strongly suggest first implementing a simple, well-written, correct, and (reasonably) performant program before attempting to optimise it further.*
- If you are in a time pinch, it should take you at most 2-3 hours to write a minimal working program and some thoughtful analysis. If you have more time and are confident in your working solution, we encourage you to elaborate on your solution to showcase your architecture and code organization skills, possibly implementing some of your ideas from the write-up questions. If you do expand on your solution after completing the basic requirements, consider using git to commit working checkpoints, in order for us to see your progress and give credit for past versions if the final version does not pan out.

More Tips:

- Unlike in many other coding challenges, your program's clarity and efficiency is more important than its optimality. Maximising profit in this case is an open problem, so we don't suggest spending a lot of time coming up with neat heuristics before actually writing a working program. (This is also why we have given you a simple heuristic that is guaranteed to assign all the loans without losing money on any loan.)
- You may use any language you like. Using a high-level scripting language will probably allow the most rapid progress. You may also use any external libraries that you wish – this will likely be helpful for parsing CSV files in some languages – but you really shouldn't need to use anything fancy (e.g. databases, multithreading, or LP solvers).