CVF Technical Take Home

In this take home, we will explore the inner workings of CVF data pipelining. There will be two parts: first we will go through the datasets that make up the back bone of CVF; second, you will be asked to build an application that can surface all the pieces that make up a CVF deal.

Some housekeeping - each ***Food for Thought*** section should be filled in with a few notes about how you’re thinking about it. This allows us to follow along to make sure you’re understanding each part correctly.

For this whole assignment, feel free to use AI / Claude Code / Codex. Also, what we are really looking for is how you handle part 2. Part 1 should only be the backbone so that you understand the concepts at CVF.

Part 1: Setting up a data pipeline for CVF (4 hours)

### **Files from the Company**

##### spend.csv

On a monthly basis, a spend file of how much the company will spend is given. The columns are defined as:

cohort - the month that the company is spending month towards

spend - the amount of spend that is happening that month

##### payments.csv

Daily, we receive a file that represents the complete transactions that a customer has spent with the company. The columns are defined as:

customer\_id - a unique identifier of the customer

payment\_date - the date that the transaction happened

amount - the transaction size

##### Exercise 1: Generating Cohorts

Above are the raw files that exist for CVF. However, at CVF, we look at these payments on a monthly basis. With that said, we will need to look at what we call a cohort\_df. The code for this is in take\_home.ipynb. This function is filled in for you.

##### cohort\_df

cohort (index) - the rounded month in which each cohort comes from

payment\_period (columns) - the month number (0-indexed) of the payments

amount (values) - the total summed amount in each of the cohort / payment\_period pairs

***Food for Thought***

1. How would you calculate the number of months before a cohort breaks even? Which cohort is closest to breaking even?
2. What is the CAC (Customer Acquisition Cost) for each cohort? How about CAC per customer # in a cohort?
3. An observation is that cohort\_df and spend\_df are separate dataframes. Should they be one? What are the tradeoffs?

### **CVF Deal Concepts**

#### predictions

CVF is in the business of predicting a company’s cohorts. Here’s the structure of how a prediction will look. There are two aspects to predictions. First, the percentage of spend in the first month that was recouped by payments. What we call the **m0** percentage. Then, what monthly **churn** looks like every month thereafter. Churn is defined to be the percentage that payments will decrease each month. For the sake of simplicity, we’ll assume that churn is equal each month. When there are actual values, we do not predict those again. We only predict from the point of the last actual value.

* m0
* churn
* scenario (enum: “WORST”, “AVERAGE”, “BEST”) - keep this in mind in part 2. It is not used in part 1.

For example, a new cohort will look something like this:

m0, m0 \* (1-churn), m0 \* (1-churn)^2, etc.

A new cohort with actual values will look something like this:

actual1, actual2, actual2 \* (1-churn), actual2 \* (1-churn)^2, etc.

#### Exercise 2: Generating predictions

Given this setup, we can now extend our cohorts to predictions. The cohort\_df now becomes predicted\_cohort\_df. In the take\_home.ipynb, please go ahead and fill in how this function will work. We provide some assertions to make sure that this is running properly.

***Food for Thought***

1. What is the LTV (lifetime value) of each cohort?
   1. How many years out should we be predicting cohorts (we defaulted to 12 in the exercise, but that’s probably too short)
2. What returns will a cohort make on their spend? What is the MOIC (Multiple on Invested Capital)?
3. In our examples, the cohort\_df only represents cohorts that are present in the company’s dataset. For example, we are missing 2020-03-01. Is this intended behavior?
   1. Empty cohort months do require spend.csv to be filled in for the month. We cannot predict without spend amount in our given structure. Do we like this?

#### thresholds.csv

payment\_period\_month

minimum\_payment\_percent

Thresholds represent a way for us to know if our companies are performing up to par. For the sake of simplicity, we represent thresholds as the **payment\_period\_month** and **min\_payment\_percent** that the payment / spend amount needs to be above. To minimize risk, we use thresholds for two purposes. 1) If a cohort goes below their limits, we collect at 100% 2) CVF can choose to stop working with the company once a threshold is breached

#### Exercise 3: Generating monthly cashflows (with thresholds) to companies

In this exercise, we will write an apply\_threshold\_to\_cohort\_df function. This function will create a boolean dataframe mask of whether or not a specific period month has failed a threshold check. This mask will be used in the next step to generate cvf\_cashflows.

**Food for Thought**

1. How does flipping to 100% on sharing\_percentage change the impact on our collections? Does it make sense why it helps mitigate risk?
2. Currently thresholds are built so that they only flip the month that we are in. Should we make it flip more than the current month?
3. Because we’re in the business of relationships, we do not simply stop working with companies the instant they go below thresholds. A question for part 2, but how do we “override” thresholds if we choose to?

#### trades.csv

Trades represent each one of the investments with the companies that we are working with. For the company, we have a **cohort\_start\_at** that represents the cohort that we are investing into. **sharing\_percentage** is the % of S&M spend that we are investing into and the % of payments we are obtaining over time. We will keep collecting until we reach our **cash\_cap** - the return amount that we are looking for.

cohort\_start\_at

sharing\_percentage

cash\_cap

#### Exercise 4: Generating monthly cashflows to companies

Now that we have trades, we can now create a get\_cvf\_cashflows\_df. This dataframe represents the amount of money that a company must pay us each month. Each trade begins with a different **cohort\_start\_at**. The **sharing\_percentage** changes if we breached a threshold or not. cash\_cap is then applied.

***Food for Thought***

1. Is CVF going to make money on this trade?
2. What are the scenarios in which we should be scared?
3. There are many levers in CVF that can be tweaked: predictions, sharing percentage, caps, etc. Any thoughts on how we should think about when to tweak each one?

# This is CVF!

At this point, you now have a working version of how CVF works. Obviously, many details are being left out but you now have a sense of the backbone.

Part 2: CVF as a product / continuously operating system (8 hours)

The above is fine and great but we have not created anything operationally of how a company sends us data, how anyone can create or update predictions or trades or thresholds.

The second part of this exercise will be to think through how one must manage all of these moving pieces. We have multiple end users here: CVF Portfolio Company, CVF Underwriting team, CVF Operations team. For this exercise, let’s focus on CVF Portfolio Company since it has both read and write operations.

### CVF Portfolio Company

Even though CVF seems simple, the structure of the deal does make it complicated. However, it is our job to make this as simple and painless to deal with on a monthly basis as we work with our portfolio companies.

At its core there are only three numbers we need from a company each month:

1. The amount of money they are going to spend in S&M next month
2. The amount of money they actually spent in the current month (we call this an adjustment)
3. The amount of money that their cohorts paid back in previous months
   1. We should make sure that sharing\_percentages that flip to 100% should be accounted for correctly
4. The payments that back up what cohorts should be paying back each month

You’re now tasked with creating an application that can, in the cleanest, simplest manner, show them the above information, and allow them to update these numbers. We want to provide the best service possible (think Michelin star quality). How can we achieve this?

To provide some structure to this:

1. How would you store all the above pieces of CVF? Database? S3 Files? Etc?
2. Is the application you’re building is a website? Internal dashboard?
3. In the application, what parts are read-only, what parts are editable? Who’s able to edit which parts?
4. The application must continuously update when pieces of data change. For example, if spend is updated, we must recalculate cashflows since our caps are different. If there are refunds on payments, we must recalculate cashflows. More examples exist but keep them in mind.

Here are some features that a company may want - this is in no particular order. This list of features might not be the full set nor the most important set:

1. What do my cohorts look like?
2. How are you going from my raw data to the cohorts? What is the methodology?
3. How much money do I owe you this month?
4. Where do I update my company’s spend numbers?
5. Are my cohorts breaking thresholds? Did we fully collect on them? Is CVF representing our terms correctly?
6. **Food for Thought:** A CVF Company does not actually care about predictions. It is our internal team that does. Does it impact any design decisions you might make?

This is left very open ended intentionally. In the world of AI, we encourage you to vibe code, use it for inspiration, or even complete parts of the exercise if you choose to.

#### How you’ll be graded:

1. Application correctness - the numbers being displayed need to be correct. We don’t want companies to question our numbers
2. Application simplicity - is it easy for companies to use our application to make the changes that they need?
3. Creativity of application - is the application built thoughtfully? Does it take on a service oriented approach?
4. Code quality - Even though the code may be written by AI, the code base should be manageable, easy to understand and most importantly, easy to maintain.
5. Ability to defend your decisions - There are many decisions and assumptions you’ve made on your way to building this application. When asked to defend, you should be able to give good reasons.

You do not need to keep any of the definitions used in Part 1. If you want to redefine dataframes to classes or redefine the content of classes, everything is fair game.

### Final Output

Please provide us the whole directory including both 1) this file with the **Food for Thought** sections filled in and 2) the notebook take\_home.ipynb filled in. On top of this, directions for how to run the application and the code required to run the application.