Problem Exploration

In general, I believe user drop-off could be caused by two main factors, which could be split into the following sub-factors:

1. The app does not meet user expectations. Which could be caused by the following sub-factors:
    1.1. There is no demand for such product.
    1.2. There is demand, but the app does not have useful content.
    1.3. There is demand, and the app does have useful content, but users cannot find it. It could be because of bad UI, bad recommendations or search engine, etc. Thus, users are still leaving it.
2. The app does meet user expectations, but they drop off it soon because:
    2.1. They read & learn useful content for them, and they stop seeing reasons to keep using it. This might be caused by different reasons: non-updating old content, or bad UI.
    2.2. A simple reason could be that the content that they needed to use the app for - is not engaging. Thus, the app ends its lifetime soon. A simple example of this could be a book. I believe the more interesting a book is, the sooner it is dropped. Because readers can't leave it before finishing. But once they read it, it's done. So, in this case, we have a problem of keeping users engaged with the app. And we need to find them.

Identifying which is the case between #1 - #2 is very important to build the right strategy. From the given information that users use the app for the first two weeks, it makes me assume that we have a problem illustrated in the 2.2 section. This could be verified by checking the users' activity, if they actively use the app during the first two weeks and then suddenly leave it. Tracking what they used the app for during this initial period could be very useful. Analysing users' activity statistics could help us identify the problem type: bad UI or non-engaging content (reference to the book example), etc.

For now, I will assume that the company has been tracking users' activity and we have data for analysis. I will assume that you provided me with data, which includes users': age,

number of days of workouts per week, and the program they spent the most time during the first two weeks. I could not imagine simpler data than this. I will generate this toy data myself to use it for analysis, and it will be skewed to support my idea that: we have 3 user clusters and they have strong preferences, which could be used for recommendations.

Solution Proposal

The data could be used to:

1. Build a simple recommendation engine for new users. New users with similar attributes/demographics could be offered with the same topics that were popular among the similar cluster group. Such engine could be used for a simple re-ordering of the "programs" list within the app's UI, or in some other way. I am going to implement a small example of such engine using the toy dataset and build a simple FastAPI app for making topic suggestions for new users.
2. We can make programs more engaging to keep users using the app by adding new AI or non-AI features. For example, we can create an AI training coach that will be guiding users 24/7 and helping them with new exercises, training methods, and recommending new diet programs. I will list all my ideas below, but I am not going to implement them today; those are fictional ideas for the future.

For case 1, I clustered users' data using a simple K-Means clustering algorithm and built a very simple topic recommender engine using FastAPI. Here is what I did:

1. Estimate how many user clusters the data contains (the K value);
2. Clustered users, checked if they have strong preferences (which was true), and saved the clustering model to reuse it via FastAPI app;
3. Wrote a simple recommender engine using FastAPI, which assigns a user to a corresponding cluster based on features: age and number of workouts per week. After knowing which cluster a user belongs to, the API returns the name of the topic/content that is the most popular among the user's cluster, based on the provided dataset statistics.

You can check my AI research code in the data_clustering.ipynb notebook file and the FastAPI app at endpoint.py.

In the second case, if our surveys showed that the content is not engaging enough for the users and thus they are leaving, in my opinion, we can add the following features:

1. An AI wellness companion assistant, with which users can share their emotional difficulties or struggles and receive professional recommendations from an AI built with corresponding knowledge.
2. AI training coach, which could create personalized training programs and guide users through training progress; This could be additionally monetized by offering specific users to buy food supplements and stuff like that;
3. To keep users bound to the app, I think adding new features like a calorie calculator and similar features related to health could be beneficial.

Trade-offs and Risks

There are huge risks associated with an AI wellness companion and an AI training coach. This includes incorrect recommendations, which may lead to training injuries or incorrect medical treatment. Thus, I think the scope of the AI usage should be well explained to users, as well as the AI has to be developed in a specific way that can prevent this from happening.

On the clustering side, the actual users may not be so well separable into clusters for recommendations as it is in my dataset, where I did this on purpose. In terms of latency, such a simple clustering model runs incredibly fast since it is not an LLM or a deep model, so we can run it on a basic CPU server without any problems. But I would perform latency analysis for larger models and would figure out the ways we could scale the system in case of high demand: spinning up new servers to balance increased load, the use of compute-efficient approaches of AI model deployment, continuous batching of requests, etc.

Optional (but encouraged)

If you had just 2 days in a discovery sprint, what would you simplify or cut?

- I would cut AI assistants. We could simply use LLM services, but testing and tackling the issues that I listed in the Trade-offs section would take more than 2 days for sure. I would

spend these 2 days trying to identify user clusters, and if those clusters do exist, then implementing such a simple recommendation engine is more doable than AI assistants. Also, risks are way smaller compared to training injuries and incorrect medical treatment.