BUSINESS
SCIENCE
CORPORATION

SOLVE · AUTOMATE · PROSPER

Role: Junior Developer
Case Study: Streamly

Candidate Brief

You have joined the data & product team at **Streamly**, a subscription video streaming service. Currently **Streamly**'s business model is that users log into their profile and they can watch whatever they want, but all suggestions on what to watch are generated based on randomly picking titles from the Streamly catalogue.

Streamly's stock price has been crashing amidst allegations that their recommendation algorithm is one of the worst in the market.

With the company facing financial ruin, you think you can prove to your boss that it's possible to slap together a recommendation algorithm over the weekend, your job depends on it!

You will work with a small, anonymised dataset to:

- Clean and model viewing behaviour in python.
- Build a simple database to which you can upload this data.
- Put together a very basic backend API that can pull data from the DB.
- Utilise the API to Visualise in a web page how well the recommendations work.
- Present your method and findings in a presentation format

You have **48 hours** from receiving the data. We expect that the objectives should take you anywhere from an hour to a full day depending on your skill level, which will leave you with plenty of time to put together a presentation.

The purpose of this Case Study is to show us, the reviewers how you analyse a problem and how you go about solving it. Be methodical in your work, document what and why you are doing things in your codebase to give us as much insight into you thought processes as possible.

Objectives and suggestions provided in this document will ensure you get to the end of the Case Study, but must be seen as the minimal possible effort required to get to completion, your final score will depend on how far you are able to expand beyond the minimum expectations.

Be sure to read through the entire briefing document before starting anything.

A Note on AI

The use of AI (Gemini, ChatGPT etc) is permitted to help you understand and solve this case study. Keep in mind that, should you progress to the panel interview, you will be questioned primarily on the solution to the case study and why you took certain design decisions and you will be expected to be able to explain your code in full.

Deliverables

- A private github repository containing all your python scripts, database generation scripts, frontend scripts as well as any supporting documentation with assumptions, analyses, jupyter notebooks etc. and a Readme.md file with any steps required to run your web application.

- We recommend that you structure your code by using separate python notebooks for each of the main Objectives, this will make it easier for you to follow along, and it will make it easier for us to score you in a structured manner.

- A presentation showcasing your method and solution, no more than six slides which includes your title slide

Business Context

Streamly offers on-demand video content (series, movies, kids' shows, etc.) in multiple countries.

There are two subscription plans:

- **Basic plan**
  - Only **1 profile** per account
- **Premium plan**
  - Up to **4 profiles** per account
  - Higher monthly price and more simultaneous streams

Each **account** can therefore represent one or more **profiles** (e.g. "Mom", "Kids", "Alex").

Streamly's current content suggestion algorithm is a random number generator that picks a number between 1 and the number of titles in the database and then suggests the corresponding title to the viewer. Survey feedback has shown that users feel like the platform never suggests anything that they want to watch.

Note: Our Director Andreas vibe coded some database improvement last week, to speed the database up he dropped all of our historical viewing statistic, whoops, he didn't think we'd need it! This means we have to start the recommendations for each profile from scratch.

Andreas says he's sorry about that, but he recommends inferring what we can from the data that we do have to figure out what recommendations to make.

Objective 1 - Data

Extract the Data

You have been provided with three data files:

titles.csv – A set of titles in the Streamly catalogue
accounts.csv – A list of primary Streamly accounts
profiles.csv – A list of account profiles

Extract them into your python workspace.

Analyse the Data

Always a good idea to see what you're dealing with, do some basic analysis of the data to see what's going on. We recommend using a **python notebook** for this step so that we can see and replay your analysis steps.

Clean the Data

No doubt you uncovered some anomalies, the gremlins in our database are always causing problems. Again, we recommend a **detailed python notebook** for this step so that we can follow your cleaning process step by step.

Please also provide a summary at the end of any issues found and explain **in one sentence for each** how you addressed the issues.

You can always return to this step later if you pick up on any data anomalies during your travels.

Objective 2 – Database

You should now have at least three sets of clean data. Next up we'll get that data into a database of our own.

Spin up your Database

For this step we recommend using a relational SQLite database to store the data, we can run it locally on your computer and it's easy to set up since it's part of the standard python library.

SQLite uses standard SQL (Structured Query Language) for CREATE TABLE, INSERT, SELECT, etc.

You can create the database directly in Python by running the following script. Once again do this in a python script in your repo so that we can recreate your DB on our side.

```
import sqlite3

# Connect (this will create streamly.db if it doesn't exist)
conn = sqlite3.connect("streamly.db")
cur = conn.cursor()

conn.commit()
conn.close()
```

Build Out Your DB

Now that you have created a streamly.db database file, the next step is to design a simple schema and create the tables needed to store your cleaned CSV data.

Keep it simple, but think about the queries you'll want to run later, for example:

"All profiles for a given account"
"All titles suitable for a given age band or kids profile"
"Titles relevant for a given country / language / preference"

Then write SQL queries to create your tables, and run them in Python to generate the tables in your streamly.db database file.

Upload your Data

Finally Upload your csv data to the relevant database tables.

Objective 3 – Recommendation Algorithm

We have our clean data in our database and we're ready to start the real work. Recommending appropriate titles to our viewers.

Read Data From the DB

Use your database connection to read back the data you need for this step into Python, you can use any format you're familiar with.

At a minimum you will likely need data for:

- Titles
- Profiles

Recommendation algorithm

Design an approach to match viewers with titles that they are likely to enjoy.

You do **not** have to build a machine learning model (although it would certainly help your prospects). We're more interested in:

- How you structure the problem,
- How you use the available data,
- How easy it will be to use your algorithm to drive recommendations in the next step.

Make sure your code for this step is reproducible and documented. Also capture an explanation on how your algorithm works in your notebook so that we can understand your logic.

DO NOT overthink this, move on to the next section once you have a useable solution you can always come back to this later to make improvements if you have time.

Test algorithm

Once you have a first pass at your algorithm:

- Inspect a few example suggestion to see if they make sense or if there are obvious mistakes.
- Adjust your logic or features if needed, and briefly note what you changed and why in a note somewhere so that we can get a sense of your iterations.

By the end of this step, you should have:

- A function that can take in a user name or id and produce a list of possible titles that they might like.

Objective 4 - Visualisation

Build a Basic Web Frontend

For this step, you should build a **very simple web page** that:

- Calls your recommendation logic (via a backend endpoint or a simple Python script)
- Shows the recommended titles visually.

You do **not** need a fancy design or framework, we'll provide some guidelines to get you started.

1. Minimal Setup
You can choose any lightweight approach you're comfortable with. For example:

- A small **Flask** app that:
  - Exposes an HTTP endpoint (e.g. **/recommendations**) that returns JSON.
  - Serves a simple HTML page that uses that endpoint.

At a high level, the structure might look like:

- **app.py** – your Flask (or similar) backend with a route that returns recommendations.
- **templates/index.html** or **static/index.html** – your frontend page.
- A bit of JavaScript in the page to call your backend and update the chart.

We are not strict about the exact structure, as long as we can run it.

2. Example Backend Shape (for Guidance Only)

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/recommendations")
def recommendations():
   # Read query parameters, e.g. age_band, country, kids_profile, seed_title_id, etc.

   # TODO: call your segmentation / recommendation logic here
   # result = get_recommendations(age_band=age_band, country=country, ...)

   result = []  # replace with real list of recommended titles
   return jsonify(result)

@app.route("/")
def index():
   # Serve your HTML page (or you can serve it as a static file)
   return "<html><body><h1>Streamly Recommendations</h1></body></html>"

if __name__ == "__main__":
   app.run(debug=True)
```

You don’t have to use this exact code, but your backend should provide **some way** for the frontend to ask for recommendations.

3. Example Frontend Shape (for Guidance Only)

Create a very simple HTML page (**index.html**) that:

- Has a small form for user input
- A placeholder **<div>** or **<canvas>** to show your chart or list to visualise recommendations.

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Streamly Recommendations</title>
 </head>
 <body>
  <h1>Streamly Recommendations</h1>

  <form id="filters">
   <label>
    Age band:
    <select id="age_band">
     <option value="18-24">18-24</option>
     <option value="25-34">25-34</option>
     <!-- etc -->
    </select>
   </label>

   <label>
    Country:
    <select id="country">
     <option value="ZA">ZA</option>
     <option value="GB">GB</option>
     <!-- etc -->
    </select>
   </label>

   <button type="button" onclick="loadRecommendations()">Get recommendations</button>
  </form>

  <div id="output">
   <!-- your chart or list goes here -->
  </div>

  <script>
   async function loadRecommendations() {
    const ageBand = document.getElementById("age_band").value;
    const country = document.getElementById("country").value;

    const resp = await fetch(`/recommendations?age_band=${ageBand}&country=${country}`);
    const data = await resp.json();

    // TODO: update your chart or list using `data`
    const out = document.getElementById("output");
    out.innerHTML = JSON.stringify(data, null, 2);
   }
  </script>
 </body>
</html>
```

Again, this is just to show the idea. You can:

- Replace the plain JSON display with any visual
- Use any charting library you like e.g. Chart.js, Plotly.js our personal favourite, Highcharts!, or even a simple HTML list.

Panicking?

Not everyone has experience building a web-based frontend. If this is your first time, don't panic.

For the purposes of this case study:

- A **very simple setup is enough as a minimum**:
    - One small Python file that runs a tiny web server (e.g. Flask),
    - One basic HTML file that calls your backend and displays the results.

- Your UI can be **extremely minimal**:
    - A couple of dropdowns or text inputs for user details,
    - A button to "Get recommendations",
    - A basic chart or even just a list of titles.

You are **not** being tested on:

- CSS skills or pixel-perfect design,
- Fancy frontend frameworks,
- Production-grade security or performance.

You **are** being tested on:

- Whether you can wire together:
    - User input → some recommendation logic → visible output,
- Whether your code and structure make it easy to understand **how** the recommendations are produced.

If you get stuck, keep it as simple as possible:

- Start with an endpoint that just returns a fixed list of titles,
- Then gradually hook it up to your actual segmentation/recommendation code,
- Then add a very basic visual on top.

By the end of this step, you should have:

- A small web page that demonstrates your recommendation logic
- At least one visual that helps us understand how your system behaves for different inputs.

Have Time to Spare?

Ask yourself:

- How could you improve your database design?
- How well did you inspect the data?
- Could you improve the segmentation Algorithm to provide better suggestions?
- Could you make improvements to the website to wow your audience?

Finally

Remember to capture your journey in a professional presentation that you will present to a second interview panel, should you proceed to the next stage.

No more than six slides which includes your title slide, we suggest including a slide on:

- Data cleaning, quirks and anomalies found etc
- Database setup and features
- Data Segmentation
- Frontend Visualisation
- Next Steps for further development