Hi!

Please find below the Solutions challenge. It's a 4 prompt challenge that will test comprehension of flows, data analysis capacity, technical troubleshooting skills, and coding competencies. Couple notes:

- Please be as detailed as possible on the approach, the analysis, the technical troubleshooting, and the simulated correspondences.
- Use Internet resources (i.e. Google) to clarify any jargon above and aid your understanding on any flows.
- Feel free to ask any clarifying questions if the initial resources aren't sufficient or if you have any questions about the prompt!
- Use excel for analysis, if possible

Best of luck!

Hey Team.

> I recently activated a Land Rover Campaign, tactic ID 343664. We're seeing delivery in our system but TripleLift numbers are roughly 4.45% higher compared to third party numbers. The pixel they are using is http://ec2-50-16-41-235.compute-1.amazonaws.com:8002/support.php.
>
> They gave me log level data for us to explore. Any chance you can take a look? Let me know if you need additional information.
>
> -Thomas

Part 1: Analysis
Okay support, time to go to work. Thomas, our Client Services ace, is having some trouble with the campaign reporting numbers. Particularly, a discrepancy between TripleLift data and a third party analytics vendor's numbers. Your job is to unblock him. You'll find the relevant dataset here https://www.dropbox.com/s/63yic0qvgb589hx/SolutionsData.xlsx?dl=0.

First the flows:

TripleLift has a javascript tag (THE JS) on the webpages of sites with which it works. When the js tag executes, it makes a request to TL servers for an ad. Let's say an ad returns. There are two entities that record that the advertisement rendered -- TripleLift will account for the impression as well as any independent verification agency.

For TL to account for the ads, TripleLift always drops a pixel that points to its own analytics collection engine[1]. This is done within THE JS that is on the site. When this occurs, TL also drops third party trackers so that clients can funnel data into their own servers. The sources for these third party trackers can be found in the tl_tactic.impression_pixel_json field (in escaped form) in the dataset given.

A. **Prompt: Illustrate the flows (on paper, google slides, etc) to depict the mechanisms and to visualize the problem.**

Now the explanation of the dataset:

- **TL Log DataSet**: Every record is a recording of a served advertisement (impression in adtech parlance) for TL tactic 343664. You'll find a rich body of data here that describes the ad rendered, the venue of the ad, and user specific information.

[1] For clarity, a 'pixel' in this usage is 1x1 hidden img element with the source set to whatever endpoint does the incrementing of the render accounting (e.g of TL pixel: <img src="//eb.3lift.com/imp/trk?id=923784" width="1" height="1">. Here the eb.3lift.com application notifies TL system of an impression event and increments the counter).

- **Client Log DataSet**: Each row represents a rendered impression for TL tactic 343664 in the client's system. As you can see, it doesn't have as much detail as TL logging but just enough to solve this problem.
- **tL_tactic**: Tactics are objects in our system that represent the creative and parameters of the advertising campaign, including the analytics scripts and pixels that need to be fired. You can find a bunch of the tactics in this spreadsheet. The tactic that Thomas is having trouble with has an ID of 343664.You can find the troublesome impression pixel by exploring this record. There are other tactics here as well to aid your understanding of what a tactic is.
- ***_mapping**: TripleLift IDs to the name of the entities of interest.

B. Prompt: Analyze this data to diagnose the source of the data misalignment and issue a correspondence to the communicate this out to relevant parties (so who are the internal or external parties? what do you tell them?).

Part 2: Exploration
Okay, now that you know where the issue is stemming from, let's explore why it's happening. TripleLift has a mechanism to force the tactics to render on the pages where our scripts are found[2]. For example, on http://makersalley.com/search/?tripleliftTest=true&tl_tactic_id=343664, you can find our unit here.

You can append the ?tripleliftTest=true&tl_tactic_id=343664 query string parameter to any URL where our tag is found. This will activate the exact code path that would be followed if this ad were organically served back to the page. As a debug hint, you can take a look at the creative payload being returned to the page by observing the /qa endpoint (http://cl.ly/1g2F2g1V2e00).

C. Prompt: Run simulations using this forced mechanism and exploring pixel drops to confirm your suspicions from part 1. Provide a clear recommendation to the relevant stakeholders / parties.

Part 3: Scripting Analysis
At this point, you might have a sense of what the issue is.

As an extra measure, let's make sure that all impressions pixels are valid by firing them off programmatically and looking for 2XX / 3XX responses. This is something we can later use on arbitrary dataset with the same schema to identify faulty trackers in the future.

D. Prompt: Write a script that will programmatically check whether all impressions pixels are valid.

[2] Make sure the TL tag is on the page first -- you should see http://ib.3lift.com/ttj?inv_code=makersalley_main on the page (or if you use a network sniffer like Ghostery, you'll catch it http://cl.ly/430c2s1N110V). Otherwise, this forcing mechanism won't work.

Details

Convert the “tl_tactic” tab to a CSV (or use this). Write a program that will

1. Ingest the CSV
2. Clean the data for analysis
3. Take each individual impression pixel from the \`impression_pixel_json\` field
    a. Fire the pixel (HTTP GET)
    b. Collect the Status HTTP response
4. Print out a summary of the results
    a. Number OK (2xx and 3xx Responses), Number Failed (4xx and 5xx Responses)
    b. List the Tactic ID & URLs that failed

Commentary

1. This does not need to be a web app (so file fetch in code / pass file as argument / read from standard input / etc)
2. Please include execution instructions, dependencies, and README
3. Keep code modular & comment your code
4. Pick language of your choice
    a. Preferred Languages: PHP, Java, Python, C, JS
    b. Bonus for Rust, Go, Scala
5. Write a unit test :)