CS 329E Final Project, due Thursday, 05/01. **Due date is not flexible**.

Ground Rules

- Choose one of three options based on your group's interests. Make sure your partner is onboard with your selection (!)
- If you are choosing option 1 or 2, construct your solution based on the guidelines provided. As this is a final project, your group is expected to work independently and drive the overall project with minimal guidance from the instructors.
- If you are choosing option 3, speak to the Prof. about your idea today and submit your proposal ASAP.
- You have only two weeks to do the work for this project, including two class periods (04/18 and 04/25).
- After you submit your project, there won't be an opportunity to resubmit for lost points due to the compressed timeline (see UT's grade reporting schedule and policies for more details).

Option 1: Fuzzy Matching

Uses Gemini, BQ, and Colab.

Methodology

So far in this course, we have used two basic techniques for matching similar entities: grouping entities by their common key using standard SQL and prompting the language model to return mappings between similar entities using its in-context learning mechanism. With the language model, we passed it a set of input entities and prompted it to map each one to a target entity or label. This is how we were able to analyze the sentiment of text. This is how we implemented fuzzy joins and how we standardized names, geo locations, etc.

In this final project, we will explore a third technique for fuzzy matching: embeddings, which are vectors or numerical representations of data in high-dimension space. Instead of prompting Gemini, we will create embeddings for all the records that need to be matched and store the resulting vectors in BQ. Once we have the embeddings we will do a nearest neighbor search on them to find the most similar entities. The advantages of searching embeddings is that it is cheaper than a token prediction approach and potentially it also potentially produces higher-quality results because the embeddings don't hallucinate.

We will evaluate the distance between each input and its nearest neighbor. If the distance is small enough, we will infer that they correspond to the same entity. We will store the resulting clusters in BQ into a vector type. We will then use the cluster information to take some action.

The action will depend on the use case, it can be joining, deduplicating, standardizing, classifying. You need to determine the appropriate action based on your problem domain and dataset.

Implementation Plan

Choose the task you want to apply this approach towards. This can be fuzzy joins, classification, standardization, deduplication, or a combination of these. Then choose your input tables from the staging layer of your warehouse. It's important that the tables come from staging because we want to compare the embeddings approach with our earlier ones which used SQL's GROUP BY or the language model's in-context learning.

Implement your solution in a Colab notebook. Be sure to register the embeddings model in BigQuery, there are some setup steps you'll need to follow for that which are detailed in the sample notebook. Make sure you use BQ's built-in functions for creating the embeddings and for running vector search, namely `ml.generate_embedding()` and `vector_search()`. Store the resulting tables from your analysis into a new BQ dataset called `fin_[your-domain]`.

Once you have found the nearest vectors, you should evaluate the resulting clusters and do something useful with them. In other words, you need to construct the next steps in the pipeline in order to solve your task.

Annotate your notebook with explanations of all the major steps so that we can follow your thought process.

Include a short conclusion that sums up your findings. The conclusion should provide a brief comparison between your previous approach and embeddings approach. Would you recommend adopting the embeddings approach into your int layer processing pipeline and if so, how?

Name your notebook `fin-[your-domain]-fuzzy-matching.ipynb` and place it into a `final-project` folder in your repo. Submit your artifacts using our normal process.

Code samples available to help you get started.

Option 2: Incremental Updates

Uses dbt, BQ, Gemini, and Colab.

Methodology

In project 6, we used dbt to orchestrate our data processing pipeline from staging to mart. However, we did not account for incremental updates to the data. What are incremental updates? These are the changes to the raw data that occur over time from all the different data sources combined; in short, they are the inserts, updates, and deletes to the raw data.

In our current implementation, if we wanted to refresh our marts with the most up-to-date raw data, we would need to re-process the entire dataset from scratch and treat it as an initial load. But this is very wasteful, especially if the number of changes to the data are small relative to the size of the entire dataset. Fortunately, the dbt designers thought of this problem and have provided us with a mechanism for processing change data called incremental models.

For this final project, you will create a new dbt project that replaces the SQL and Python models that you wrote for project 6 with incremental materialization (`materialized='incremental'`). You will incorporate the `is_incremental()` macro into your models to tell dbt which rows have changed and need processing. You will specify a unique key per row so that dbt can handle updates to existing data in addition to inserts.

Implementation Plan

Read through the dbt documentation to understand how incremental models work.

Make a copy of your dbt project folder from project 6 and call it `fin-[your-domain]`. Create a new dbt profile for this project and update the target datasets in `dbt_project.yml`. Append the prefix `fin-` to your dataset names so as not to overwrite the ones you made in project 6.

Update the model files (SQL and Python) with the incremental materialization strategy and unique key for each model. Note that the unique key can be made up of multiple fields as shown here.

Implement the `is_incremental()` macro for each model. In Python, it’s `dbt.is_incremental` as shown here.

Execute `dbt run` to process the initial load. The results from this load should be nearly identical to the state of your final tables in project 6 (except for the randomness introduced by the LLM).

Prepare an `incrementals` folder in your bucket and populate it with some change data for each source. Note: if your data sources haven't changed since project 1, you can create some fake data to simulate new and changed data.

Prepare a Colab notebook that ingests the change data into your raw tables. Make sure that the `_load_time` value for your change data is greater than the original data (April `25 versus January `25).

Execute `dbt run` to process the change data from your raw tables. If your implementation is correct, dbt should activate the `is_incremental()` macro for each model and you should see only the changed records being propagated into the staging, int, and mart tables. Hint: if you end up with duplicate records in any of your tables, you have done something wrong!

There are a few code samples available from last semester, but they contain some additional complexity in the form of snapshot tables which are beyond the scope of this project.

For submission, please follow the same format as Project 6.

Option 3: Choose your own adventure

If your group would rather work on a different idea, please write up a short proposal and email it to the instructors. We will review and let you know if it's approved. If you intend to choose this option, you should speak to the Professor about your idea today even if you have not worked out all the details yet. Given the short timeline of this project, please propose something that's doable in two weeks time. Ideally, it should be something that's fun to explore and a natural extension to the artifacts you have built in this course.

Your proposal should be submitted no later than **Sunday, April 20th.** Any submissions received after that date will not be reviewed.