Unacast report

1. Key findings

- **Duplicate Rows**
  - Some files contain duplicate rows. These duplicates are handled by retaining only the first occurrence of each duplicate row.
- **Venue Types**
  - The variable `venue_type` includes three unique categories: *university*, *coffee shop*, and *cinema*.
  - For some observations, the `venue_type` is recorded as "unknown," but the `venue_id` clearly maps to one of the categories:
    - **University**: `UN000`, `UN001`, `UN002`, `UN003`, `UN004`
    - **Coffee Shop**: `CO000`, `CO001`, `CO002`, `CO003`, `CO004`
    - **Cinema**: `CI000`, `CI001`, `CI002`, `CI003`, `CI004`
  - Missing values in the `venue_type` column were also filled based on the mapping of `venue_id` to its respective category.
- **Visitor Timestamp**
  - For some observations, the `visit_start_time` was greater than the `visit_end_time`, which is not possible. To resolve this, the values were swapped to ensure correctness.
  - Missing values in the `visit_end_time` column were filled using the **median time spent** by visitors, calculated from the entire dataset.
  - The median was chosen because the data is right-skewed, indicating that a small proportion of visitors spend significantly longer times at venues.
- **Venue Information**
  - Each file contains information for all 15 venues, with the exception of the file for `2024-11-13`, which lacks visitor data for the coffee shop `CO000`.
- All the venues belong to same category show similar visitor count patterns everyday. As the task is to forecast `total_visitor_count` for next day for each venue, this problem is considered as multiple time series forecasting problem and `venue_id` is important to capture venue specific patterns.
- **Training info**
  - Data from `2024-10-28` until `2024-11-11` is considered as train and test data. This data is used to train **Lightgbm**. Lag values and date related values are extracted from each file and used as features to train the model. The `venue_id` is uses as a categorical variable to capture venue specific patterns. All the files from `2024-11-12` are treated as daily data and created forecast for next day based on all the previous historical data. Incremental training method is used to update the model with new data daily.

2. Questions

How would you scale this pipeline to handle:

- **20 million locations of 20 million users**
  - To scale the pipeline for pre-processing, feature engineering, model training, and prediction, I would use Google Dataflow as it creates a DAG and uses worker autoscaling for dynamic resource allocation. For model training and updates, Vertex AI can be used to orchestrate the distributed training of the model.
- **Real-time updates**
  - To predict the total visitor count forecast on real-time data, Pub/Sub and the streaming mode in Dataflow can be useful to process and deliver the data in real time.
  - I’m not completely familiar with real time updates and vertex AI and would be interested to work on this in future.

How would you store the data?

- I would use BigQuery to store the data in a partitioned table based on `date`, as this makes it easy to access the data for forecasting across all venues. Depending on the specific use case, the dataset can also be further partitioned or clustered by `venue_id`

How would you monitor data quality?

- I would use tools like Great Expectations to monitor data quality. It allows you to create custom test cases to validate that the data meets predefined standards before storing into BigQuery. Additionally, it provides the ability to set alerts when data fails to meet these expectations.

What would your daily orchestration look like?

- For daily orchestration, as soon as the data file is uploaded to the GCP bucket, a Dataflow job is triggered to pre-process the data, create features, and store the results in BigQuery. Next, Vertex AI is triggered for model training or updates. Later, another Dataflow job is used to compute the predictions.