**End-to-End Data Engineering Pipeline for Uber NYC Taxi Data**

**Introduction**

In this project, I developed an end-to-end data engineering pipeline using Mage AI, Google Cloud Platform (GCP), and BigQuery to process and analyze the NYC Uber taxi dataset. The pipeline handles data ingestion, transformation, and loading (ETL) into a centralized data warehouse and ultimately visualizes the insights using Looker Studio. The process is hosted on a GCP Virtual Machine (VM) for scalable and cloud-native deployment.

**Dataset Used**  
TLC Trip Record Data Yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

Here is the dataset used in the video - <https://github.com/darshilparmar/uber-etl-pipeline-data-engineering-project/blob/main/data/uber_data.csv>

More info about dataset can be found here:

1. Website - <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>
2. Data Dictionary - <https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf>

**Tools & Technologies Used:**

* Mage AI
* Google Cloud Platform (VM, IAM, BigQuery, Cloud Storage)
* Python & Pandas
* Looker Studio

**ETL Pipeline Design in Mage AI**

Mage AI, a modern data orchestration platform, was used to build a modular and maintainable ETL pipeline. I structured the project into three clearly defined phases: Extract, Transform, and Load, leveraging Mage’s visual and code-based block system to manage each stage effectively.

**Extract Phase**

In the extract phase, I created a Data Loader block in Mage AI to ingest the raw dataset. The dataset was originally stored as a CSV file and included comprehensive information on NYC taxi trips, such as timestamps, trip distances, passenger counts, rates, fare amounts, and geolocation coordinates (latitude and longitude).

To ensure flexibility, the block was designed to support file ingestion both from the local file system and from a Google Cloud Storage (GCS) bucket. For this, Mage AI’s environment variables and cloud credential handling were configured appropriately using io\_config.yaml. Once connected, the data was loaded into a structured Pandas DataFrame for downstream processing.

**Transform Phase**

This phase was the core of the ETL pipeline, where raw trip data was cleaned and transformed into a star schema format to support analytics and dashboarding.

I created a Transformer block within Mage AI to handle all transformation logic. The block was designed to break down the flat dataset into a fact table and multiple dimension tables, a structure highly suited for querying in BigQuery and BI tools.

Key steps in the transform phase included:

* Datetime Conversion: The pickup and dropoff timestamps were first converted to proper datetime formats to enable time-based aggregations and slicing.
* Dimension Table Construction:
  + datetime\_dim: Included fields like hour, day, month, year, and weekday for both pickup and dropoff times.
  + passenger\_count\_dim: Captured the number of passengers per trip and mapped unique identifiers.
  + trip\_distance\_dim: Represented various distances covered in trips with unique IDs.
  + rate\_code\_dim: Translated rate codes into human-readable categories (e.g., JFK, Standard Rate).
  + pickup\_location\_dim & dropoff\_location\_dim: Geospatial data (latitude and longitude) was used to define unique location entries using composite keys.
  + payment\_type\_dim: Mapped payment codes to readable types (e.g., Credit card, Cash, etc.).
* Fact Table Construction:  
  The central fact table referenced all dimension tables through foreign key-like fields and held transactional data such as fare\_amount, tip\_amount, mta\_tax, and total\_amount.
* Data Integrity: To avoid redundancy and ensure efficient joins, I applied deduplication strategies, composite key generation, and indexing during this transformation stage.

The transformed data was finally stored as a dictionary of tables in memory, ready for export.

**Load Phase**

For the load phase, I developed separate Data Exporter blocks in Mage AI for each individual table—both the fact table and all the dimension tables.

Each exporter block was configured with:

* A specific table\_id targeting a table in Google BigQuery (e.g., uber\_dataetl.datetime\_dim, uber\_dataetl.fact\_table, etc.).
* A config file loader pointing to the io\_config.yaml file, which included authentication credentials and project configurations.
* Load parameters such as if\_exists='replace', ensuring that re-runs would overwrite existing tables for consistency.

Mage AI managed the interaction with BigQuery using the built-in BigQuery integration and automatically handled the creation of tables and schemas based on the DataFrame structure.

Once exported, each table was visible in the BigQuery console under the dataset uber\_dataetl. This schema is optimized for analytics.

**Export and Verification**

Using Mage AI’s modular Data Exporter blocks:

* I **loaded all eight tables**—seven dimension tables and one fact table—into a BigQuery dataset named **uber\_dataetl**.
* Each block was configured to overwrite the existing table (if\_exists='replace') to ensure that any changes in transformation logic were reflected in BigQuery immediately.
* After successful export, I **verified the dataset schema** directly in the BigQuery console. Each table appeared with the correct schema definitions and column types as inferred from the Pandas DataFrame.

The star schema in BigQuery allowed me to perform fast and expressive SQL queries, joining fact and dimension tables as needed to derive meaningful analytics. For example:

* **Trip frequency analysis** by joining fact\_table with datetime\_dim to break down ride counts by hour or weekday.
* **Revenue analytics** by aggregating total\_amount or fare\_amount across different geographies and time ranges.
* **Passenger behavior** insights by correlating passenger count with tip amounts or trip distances.
* **Geospatial trends** by filtering or grouping based on pickup and dropoff coordinates.

BigQuery’s serverless architecture and SQL engine allowed me to run these analytical queries on large volumes of data with high performance and low latency.

**Interactive Dashboards with Looker Studio**

To turn the raw analytical power of BigQuery into visually compelling insights, I integrated **Looker Studio (formerly Data Studio)** with my BigQuery dataset.

**7.1 Connecting to BigQuery**

* I connected Looker Studio directly to the **uber\_dataetl** dataset stored in BigQuery.
* Since the data was already organized into a clean star schema, I was able to **define data sources** and set relationships between fact and dimension tables intuitively.

**7.2 Dashboard Development**

With Looker Studio’s drag-and-drop interface, I created a series of **interactive and insightful dashboards**. These dashboards empowered users to explore the data dynamically and uncover trends without writing any SQL.

Key visualizations included:

* **Trip Volume Trends**:
  + Line charts showing the number of trips by **day of week** and **hour of the day**.
  + Heatmaps displaying hourly patterns across weekdays, identifying peak commuting hours.
* **Fare and Revenue Insights**:
  + Box plots and histograms analyzing **fare amount distributions**.
  + Time series showing **revenue trends** over days, months, or across holiday seasons.
* **Geospatial Distribution**:
  + Geographic maps using **pickup and dropoff coordinates** to highlight high-density zones.
  + Filters to zoom in on specific boroughs or neighborhoods.
* **Payment Type Analytics**:
  + Pie charts and bar charts comparing usage of **credit cards**, **cash**, and other payment methods.
  + Trendlines showing how payment methods vary by time of day or fare size.
* **Passenger Behavior**:
  + Cross-tabulations of **passenger count vs. tip amount**, and **trip distance correlations**.

**7.3 User Interactivity**

All dashboards were enhanced with **interactive filters**, including date range selectors, dropdowns for rate codes, sliders for trip distance, and segmented views based on time or location. This allowed end users to **slice and dice the data** according to their individual analysis needs.

**Final Outcome**

With the successful integration of **Mage AI**, **Google BigQuery**, and **Looker Studio**, I built a robust end-to-end data engineering pipeline. The resulting system supports:

* Clean and query-optimized data ingestion
* Real-time analytics using BigQuery SQL
* Business-friendly dashboards through Looker Studio

This project demonstrates a complete data engineering workflow—from raw data to actionable insights—using Mage AI and GCP services. The modular and cloud-native approach ensures scalability, transparency, and reproducibility. By integrating with Looker Studio, the data was transformed into a usable business intelligence dashboard for analytical decision-making.