**Kalvad Data Engineer Test – Mahmoud Ayman Kharoof**

Duration: 1 hour

**Dataset**

Use the following open data set:

* **NYC Yellow Taxi Trip Records** (available [here](https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page))
  + Select a single month of data for the test to keep processing time manageable.
  + The dataset includes information such as pickup and drop-off times, trip distances, fare amounts, and payment types.

**Part 1: SQL Challenge (20 minutes)**

Using the dataset, complete the following tasks:

**Before starting this task, I converted the data to csv as per the instruction provided by NYC Government website, which is using python to convert:**

**import pyarrow.parquet as pq**

**trips = pq.read\_table('yellow\_tripdata\_2024-01.parquet')**

**trips = trips.to\_pandas()**

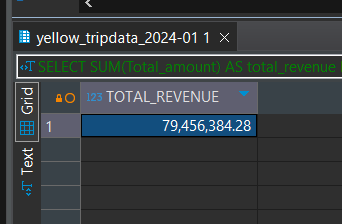
**# Export to CSV**

**trips.to\_csv('yellow\_tripdata\_2024-01.csv', index=False)**

1. **Total Revenue Calculation**: Write an SQL query to calculate the total revenue generated by trips that ended within the last 30 days of the data.

**SELECT** **SUM**(Total\_amount) **AS** *total\_revenue*

**FROM** "yellow\_tripdata\_2024-01"



**As shown the Total Reevnue for NYC Yellow taxi in January 2024 was 79,456,384.28 $**

1. **Top 3 Pickup Locations**: Find the top 3 pickup locations by total revenue over the last 30 days in the dataset.

**The correct query is, to fetch the Top 3 Pickup Locations in January**:

**SELECT** PULocationID, **SUM**(Total\_amount) **AS** *revenue*

**FROM** "yellow\_tripdata\_2024-01"

**GROUP** **BY** PULocationID

**ORDER** **BY** *revenue* **DESC**

**LIMIT** 3;

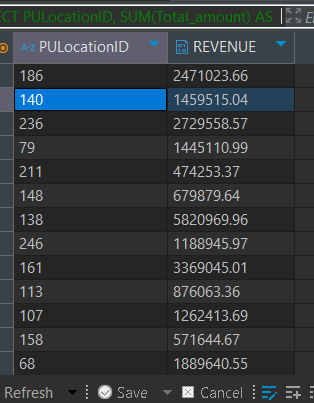
**Due To memory limitation in my system, I did the following:**

1. **Executed this query, to cover all location and their revenue:**

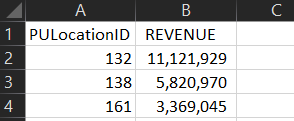
**SELECT** PULocationID, **SUM**(Total\_amount) **AS** *revenue*

**FROM** "yellow\_tripdata\_2024-01"

**GROUP** **BY** PULocationID;



1. **Copied the table pasted in excel, sorted by revenue to get the top 3 Locations by Idin January are:**



1. **132**
2. **138**
3. **161**

**I would assume 132 is** [**Empire State Building**](https://en.wikipedia.org/wiki/Empire_State_Building)**, 138 is**[**Ellis Island**](https://en.wikipedia.org/wiki/Ellis_Island)**, and 161 the Statue of Liberty, since they are the busiest places according to google.**

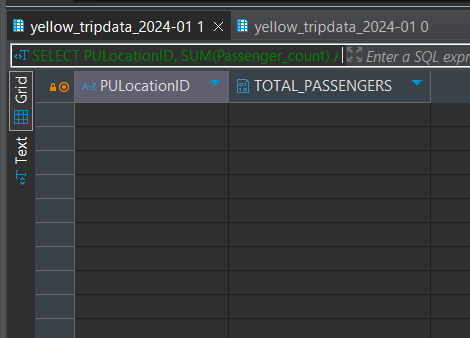
1. **Frequent Riders**: Identify passengers (if available) who completed more than 5 trips within the last 30 days of the data.

**SELECT** PULocationID, **SUM**(Passenger\_count) **AS** *total\_passengers*

**FROM** "yellow\_tripdata\_2024-01"

**GROUP** **BY** PULocationID

**HAVING** *total\_passengers* > 5;



**According to the provided output, there are no Locations with more than 5 passengers in January in NYC for yellow taxis**

*Note:* You may assume that any location data, like pickup location, is defined by the columns in the dataset (e.g., pickup\_location\_id).

**Part 2: Data Pipeline Design (20 minutes)**

Describe a data pipeline that:

1. **Ingests**: Downloads the NYC Yellow Taxi Trip Records for a specified month from the provided URL.
2. **Transforms**: Filters the dataset to include only trips with fares over $10 and renames columns to align with the following schema:
   * trip\_id, pickup\_time, dropoff\_time, pickup\_location, dropoff\_location, fare\_amount
3. **Loads**: Inserts the transformed data into a PostgreSQL or TimescaleDB database.

In your description, include:

* The tools you would use (e.g., Airflow, Python scripts).
* How you would manage any download or connectivity issues.
* Any data quality or governance measures you would consider.

**Overview of the Pipeline**

**This code sets up a data pipeline using Apache Airflow to automate three main steps for managing and processing NYC Yellow Taxi data:**

1. **Download: Retrieves NYC Yellow Taxi data for a specific month from a given URL, which is obtained from the Network Tab after downloading the data from the NYC government website.**
2. **Transform: Cleans and prepares the data by filtering and renaming columns to match the target schema.**
3. **Load: Inserts the cleaned data into a PostgreSQL database in batches, ensuring efficient handling of large datasets.**

**How Each Step Works in Detail**

1. **Tools and Libraries Used:**
   * **Apache Airflow: Manages the pipeline, including scheduling, dependency management, task retries, and error notifications.**
   * **Python Libraries:**
     + **Requests: For downloading data files from external URLs.**
     + **Pandas and PyArrow: For data manipulation and transformation. PyArrow is used to read the .parquet files, which is recommended by the “Handling .parquet from the NYC government”, while Pandas enables data filtering, renaming, and quality checks.**
     + **SQLAlchemy: Provides a connection to PostgreSQL and enables efficient batch insertion.**
     + **Logging: Custom logging is implemented to capture and report on pipeline events and errors, aiding in debugging and monitoring.**
2. **Download and Error Handling:**
   * **Function: The download\_data() function handles downloading the file using the requests library. The URL is dynamically generated based on the user-defined year and month parameters.**
   * **Error Handling:**
     + **If the download fails (e.g., due to network issues or invalid URL), an exception is raised.**
     + **Airflow’s retry mechanism is configured to retry the task up to three times with a 5-minute delay between retries. This mitigates temporary network failures.**
   * **Logging: Success and error messages are logged at each step to provide insights into the download process. If the download fails permanently after retries, a custom error message is logged for easy troubleshooting.**
3. **Data Quality and Transformation:**
   * **Function: transform\_data() reads the downloaded .parquet file into a pandas.DataFrame using pyarrow, then processes the data by applying several quality checks and transformations.**
   * **Transformations:**
     + **Filtering: Only trips with fares above a user-defined threshold (default: $10) are retained.**
     + **Renaming Columns: Columns are renamed to a consistent format to align with the target schema (e.g., renaming tpep\_pickup\_datetime to pickup\_time).**
   * **Data Quality Checks:**
     + **Null Value Removal: Rows with nulls in critical columns (e.g., pickup\_time, dropoff\_time, fare\_amount) are removed.**
     + **Logical Consistency Checks: Ensures that pickup\_time is before dropoff\_time and that fare\_amount is positive.**
     + **Logging: Logs the count of rows that passed and failed the checks, helping identify potential data quality issues in the source data.**
   * **Output: The cleaned data is saved to a CSV file (transformed\_trips.csv) for efficient batch loading in the next step.**
4. **Loading Data into PostgreSQL:**
   * **Function: load\_data() reads the transformed CSV file in chunks and loads it into PostgreSQL to prevent memory overload.**
   * **Batch Insertion: Using SQLAlchemy, the function loads data in chunks of 10,000 rows, allowing efficient handling of large files and preventing memory issues.**
   * **Database Connection:**
     + **A SQLAlchemy engine is created to connect to PostgreSQL using the psycopg2 driver.**
     + **Once the data is loaded, the engine is closed to release database resources.**
   * **Error Handling and Logging:**
     + **If a batch fails to load, an error is logged, and the task retries according to Airflow’s configuration.**
     + **Logs the number of rows loaded in each batch, providing visibility into the data load process.**

**Airflow Configuration**

* **User-Defined Variables:**
  + **Dynamic Configuration: The pipeline uses Airflow’s Variable feature to allow users to configure key settings (year, month, minimum fare amount, and notification email) without modifying the code. This approach makes the pipeline flexible and adaptable to different data sources and requirements.**
  + **Parameters Used:**
    - **taxi\_data\_year and taxi\_data\_month: Define the dataset’s year and month.**
    - **min\_fare\_amount: Sets the minimum fare filter threshold.**
    - **notification\_email: Specifies the email address for failure notifications.**
* **Retry and Notification Setup:**
  + **Retries: Each task (download, transform, load) is set to retry up to three times with a 5-minute delay between attempts. If a task ultimately fails, an email notification is sent to the user-defined email address.**
  + **Logging and Monitoring: Detailed logging is used throughout each function to capture task-specific information, errors, and data processing metrics. Airflow’s UI provides a view of task states and logs for each run, aiding in monitoring and debugging.**

**Part 3: Data Processing with Python (Optional)**

Write a Python script to:

1. Download the specified month's dataset if it is not already available locally.
2. Connect to a TimescaleDB or PostgreSQL database and load the data.
3. Calculate the average fare per day of the week for the specified month.
4. Generate a summary report listing each day of the week and the corresponding average fare, saving it to a CSV file named average\_fare\_per\_day.csv.

**My approach:**

**Overview**

**This Python script automates a data processing workflow in four main steps:**

1. **Download the Data if it’s not already saved locally.**
2. **Load the Data into a PostgreSQL Database so it can be queried and analyzed.**
3. **Calculate the Average Fare for Each Day of the Week in the specified month.**
4. **Save the Results as a CSV file for easy reference.**

**Each function in the script performs one of these steps.**

**Step-by-Step Explanation**

1. **Download the Data (download\_data):**
   * **The script checks if the data file for the specified month (yellow\_tripdata\_2024-01.parquet) is already in the folder.**
   * **If not, it downloads the file from a public URL and saves it locally.**
   * **If the file is already there, it skips the download to save time and bandwidth.**
2. **Load Data into PostgreSQL (load\_data\_to\_db):**
   * **The script connects to a PostgreSQL database using the provided database details (like database name, username, password, etc.).**
   * **It reads the data from the downloaded .parquet file using pyarrow and pandas.**
   * **It then writes this data into a PostgreSQL table (nyc\_taxi\_trips), replacing any existing table with the same name.**
3. **Calculate the Average Fare per Day of the Week (calculate\_average\_fare):**
   * **The script reconnects to the PostgreSQL database.**
   * **It runs a SQL query that calculates the average fare for each day of the week (Sunday to Saturday) for the specified month (January 2024).**
   * **It extracts and groups data based on the "day of the week" and averages the fares.**
   * **The result is stored in a pandas.DataFrame, where the day numbers (0 to 6) are mapped to day names (Sunday to Saturday).**
4. **Save the Results to a CSV File (save\_to\_csv):**
   * **The calculated average fares are saved in a CSV file named average\_fare\_per\_day.csv.**
   * **This file can then be used for further analysis or reporting.**

**Running the Script (\_\_main\_\_ Section)**

* **When you run this script, it goes through each of the steps:**
  + **Downloads the data (if needed).**
  + **Loads it into the database.**
  + **Calculates average fares by day of the week.**
  + **Saves the final report to a CSV file.**

**If there are any errors during the process, they are caught and printed, making it easy to identify issues.**

**After getting the data I found:  
1. Average Fare Across the Week:**

* **The overall average fare for the week is approximately $18.18.**

**2. Day-to-Day Fare Variations:**

* **The day with the highest average fare is Monday, with an average fare of $19.36.**
* **The lowest average fare occurs on Saturday, at $16.97.**
* **This suggests slightly higher fares on weekdays, particularly Mondays, with lower fares during the weekend, possibly due to demand patterns.**

**3. Consistency of Fares:**

* **The standard deviation of fares is around $0.71, indicating that fares remain fairly consistent throughout the week with only minor fluctuations.**

**4. Weekly Fare Trend:**

* **Fares are fairly stable midweek (Tuesday to Thursday, all averaging around $18.15 to $18.28).**
* **The slight increase on Monday could indicate a higher demand for taxi services at the start of the workweek.**

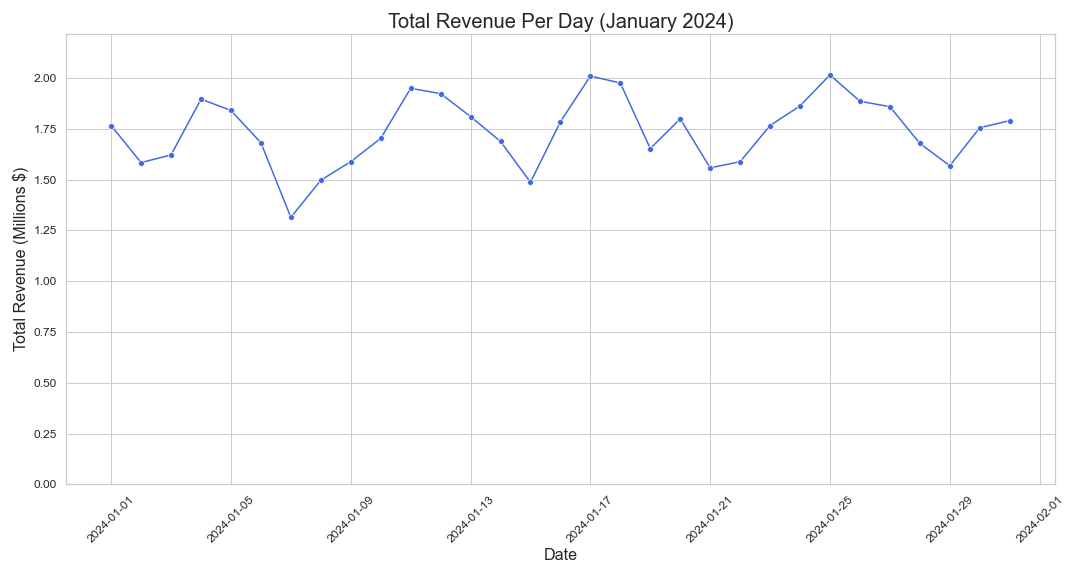
**Part 4: Data Visualization (20 minutes)**

Using the dataset, display some meaningful information by creating a chart. Choose one of the following visualization options:

1. Revenue Over Time: Display a time series chart showing the total revenue per day.
2. Top Pickup Locations: Show a bar chart of the top 5 pickup locations by total revenue.
3. Trip Distances: Display a histogram of trip distances.

*Instructions*: Take a screenshot of your visualization and include it in the repository. Name the file data\_visualization.png.

After I visualized the Revenue over Time I cam up with the following analysis:



analysis of the **Total Revenue Per Day** graph for January 2024:

1. **Daily Revenue Fluctuations**:
   * Revenue fluctuates throughout the month, with daily values ranging between approximately **$1.25 million** and **$2.0 million**.
   * This variation could reflect changes in demand, possibly due to factors like weather, events, or typical weekday vs. weekend patterns.
2. **Periodic Dips and Peaks**:
   * There are notable dips around **January 6th and January 12th**, where revenue drops to the lower end of the observed range.
   * Peaks occur around **January 3rd, January 13th, January 17th, January 25th, and January 29th**, with revenue nearing or reaching **$2 million**. This could indicate higher demand on these days, which might be weekends or specific events drawing more ridership.
3. **Revenue Recovery After Dips**:
   * After each low point, the revenue consistently rises, suggesting demand may be rebounding on specific days following lower-than-average demand. For example, the drop around **January 12th** is followed by an increase around **January 13th** and beyond, peaking again around **January 17th**.
4. **End-of-Month Stability**:
   * Towards the end of January, revenue appears more stable, with smaller fluctuations. This might indicate that weekday and weekend patterns are balancing out.
5. **Possible Day-of-Week Pattern**:
   * The recurring peaks and dips might reflect weekday vs. weekend trends, where certain days consistently see higher or lower revenue. This would need further analysis, but if weekends have higher ridership, the revenue peaks might align with Saturdays and Sundays.
6. **Overall Trend**:
   * While daily revenue fluctuates, there’s no clear upward or downward trend across January, suggesting a relatively stable demand pattern for taxi services within this month.

**Submission**:

* Publish your work in a GitHub repository (or any other Git repository).
* Ensure the repository includes all code files, SQL queries, and a README.md with setup instructions.
* Share the link to your repository upon completion.