Public Transport Demand Aggregation

Dhanyamol Devassy

Contents

Executive Summary

This project focuses on building a cloud-based, automated data pipeline for analyzing public transport demand in relation to external factors like weather.

The pipeline integrates raw NYC taxi trip data with historical weather data to generate actionable insights for demand forecasting.

Key AWS services used: S3, Glue, Athena, QuickSight, and Airflow for orchestration.

Business Impact:

- Helps transport agencies identify demand peaks and allocate resources effectively.

- Enables weather-adjusted demand forecasts for ride-hailing and taxi fleet operators.

- Provides scalable architecture suitable for other cities or datasets.

Introduction

Urban transport demand fluctuates based on multiple factors including time of day, day of week, and weather.

Traditional static planning often leads to inefficiencies like over- or under-supply of transport services.

Objective:

- Build a data pipeline that ingests raw taxi trip data.

- Clean and aggregate data by time and location.

- Enrich transport data with weather factors such as temperature, humidity, and wind speed.

- Store processed datasets for querying and visualization.

- Enable business dashboards for decision-making.

This project demonstrates a professional-grade ETL + analytics pipeline using modern cloud services.

Architecture Overview

Architecture Components:

Service	Purpose in Project
Amazon S3	Storage for raw taxi data, weather data, and processed outputs. Acts as a central data lake.
AWS Glue	Serverless ETL for transforming raw taxi trip data and enriching it with weather features.
AWS Glue Crawlers	Automatically detect schema and create tables in the Glue Data Catalog for processed datasets.
Amazon Athena	Serverless SQL query engine to analyze transformed/joined datasets directly from S3.
Amazon QuickSight	Business intelligence tool to build interactive dashboards for demand forecasting and analysis.
Apache Airflow (Docker)	Orchestrates the entire pipeline: ingestion → Glue ETL → Crawlers → Athena validation → dashboards.

Data Flow:

- Raw CSV files uploaded to S3.

- Glue jobs transform into hourly aggregates and join with weather.

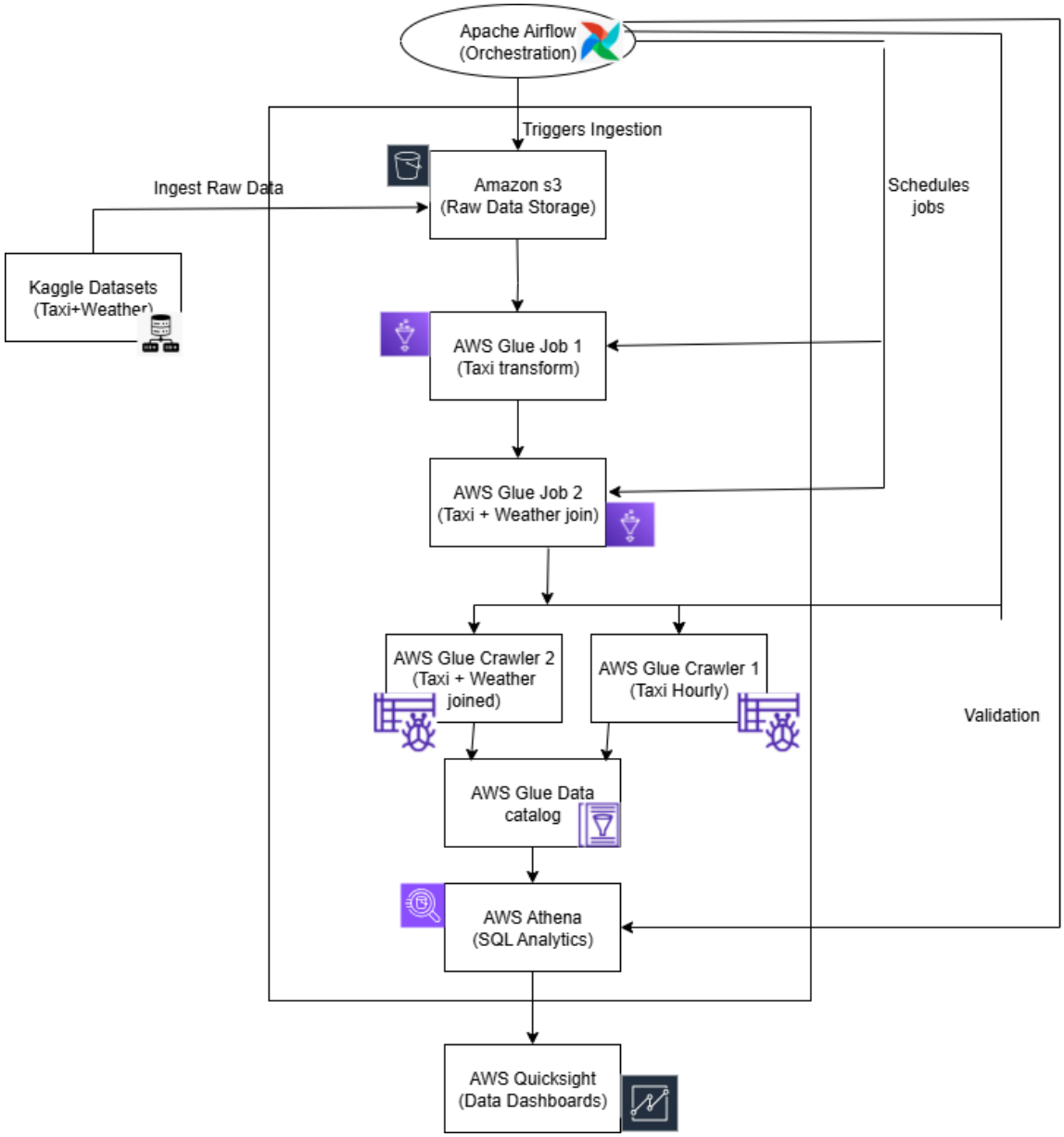
- Crawlers register schemas in Glue Data Catalog.

- Athena queries analyze processed datasets.

- QuickSight dashboards visualize trends.

- Airflow DAG orchestrates ingestion → ETL → validation.

Architecture Diagram



Datasets

1. NYC Yellow Taxi Dataset:

- Source: NYC Taxi & Limousine Commission (Kaggle mirrored).

- Key Fields: pickup_datetime, dropoff_datetime, passenger_count, trip_distance, fare_amount, etc.

- Time coverage: Historical data spanning multiple years.

2. Weather Dataset:

- Historical weather (temperature, humidity, pressure, wind speed).

- Granularity: Hourly measurements.

- Joined with taxi trips by date + hour.

Schema Examples:

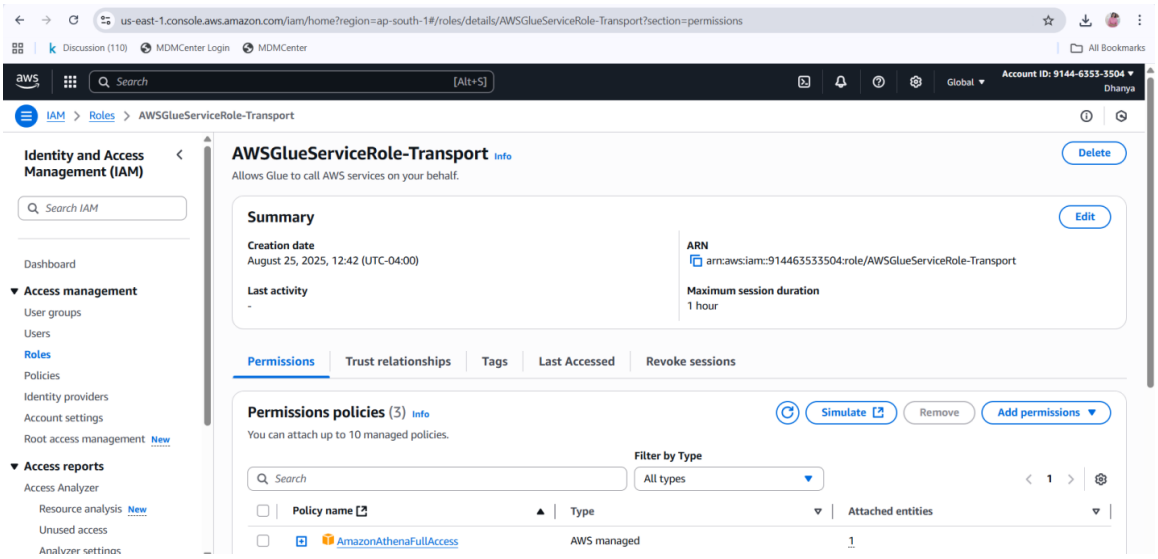
- Taxi raw CSV: [VendorID, pickup_datetime, dropoff_datetime, passenger_count, trip_distance, fare_amount, ...].

- Weather CSV: [datetime, temperature, humidity, pressure, wind_speed].

Environment Setup

1. IAM Role:

- Created `AWSGlueServiceRole-Transport` with permissions for Glue, S3, and Athena.



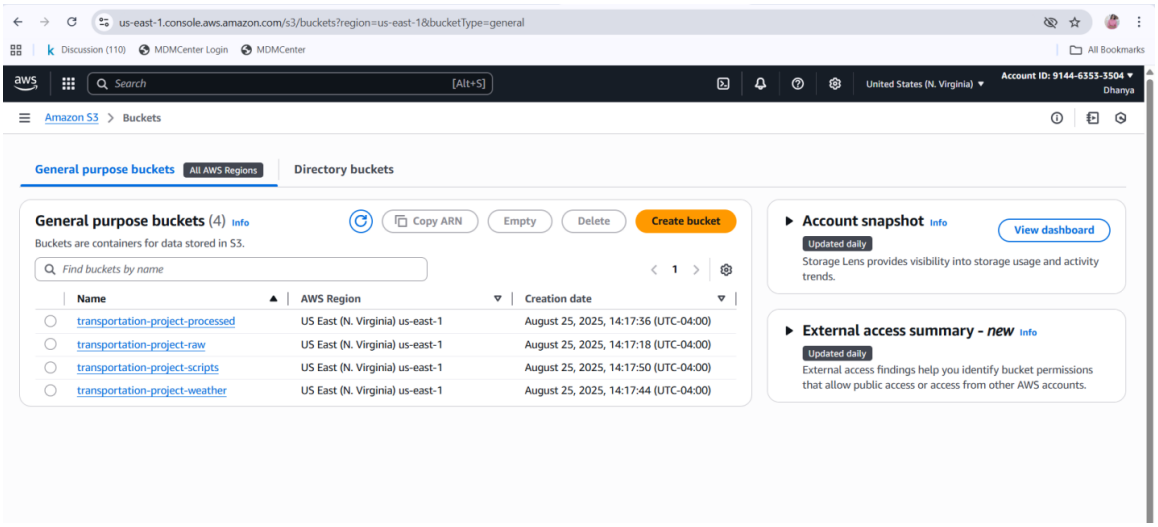
2. S3 Buckets:

aws s3 mb s3://transportation-project-raw --region us-east-1

aws s3 mb s3://transportation-project-processed --region us-east-1

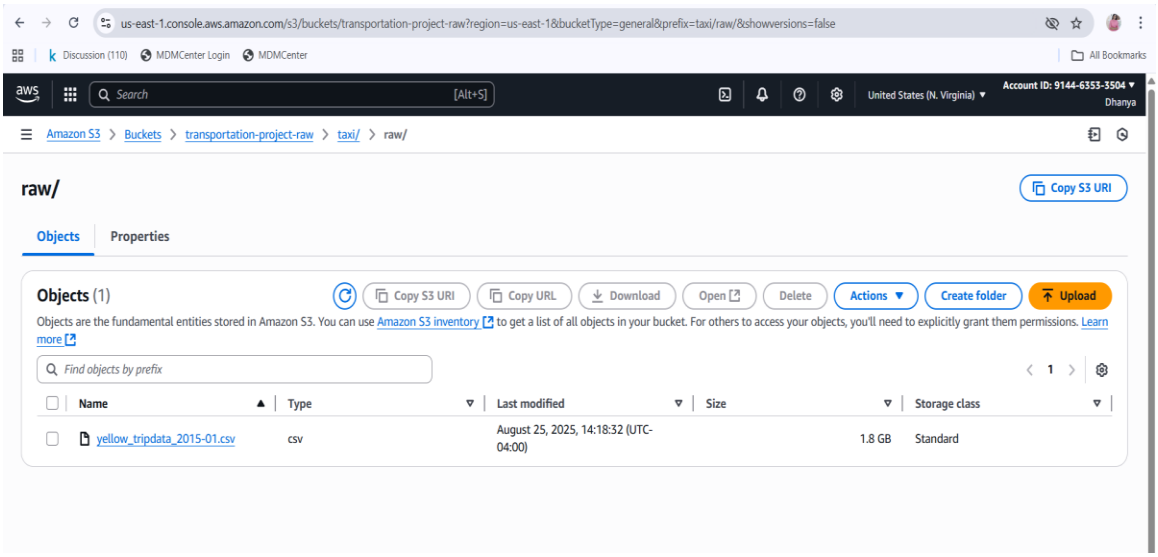
aws s3 mb s3://transportation-project-weather --region us-east-1

aws s3 mb s3://transportation-project-scripts --region us-east-1



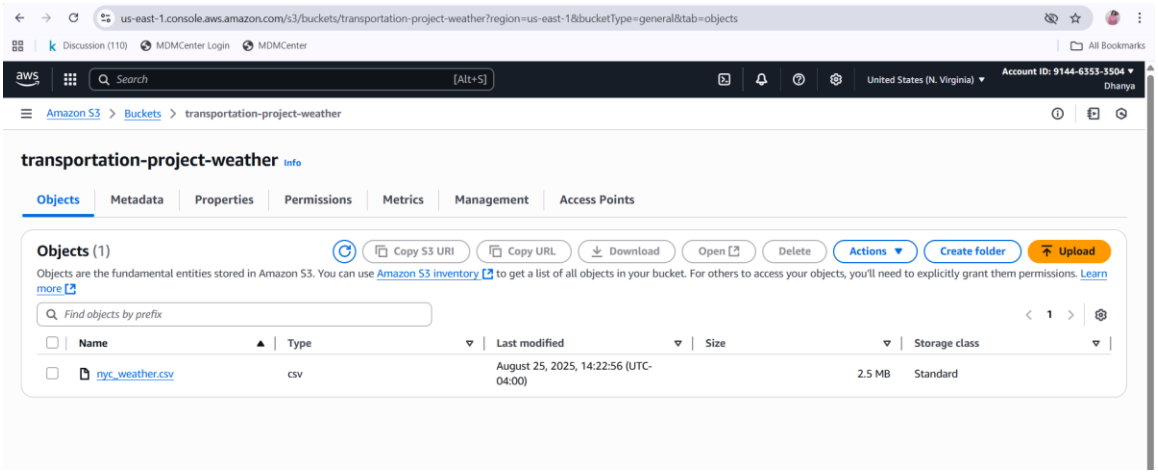
- transportation-project-raw → stores raw taxi data.

```
D:\public-transport-demand>aws s3 cp data/raw/yellow_tripdata_2015-01.csv s3://transportation-project-raw/taxi/raw/
upload: data\raw\yellow_tripdata_2015-01.csv to s3://transportation-project-raw/taxi/raw/yellow_tripdata_2015-01.csv
```



- transportation-project-weather → stores weather CSVs.

```
D:\public-transport-demand>
D:\public-transport-demand>aws s3 cp data/weather/nyc_weather.csv s3://transportation-project-weather/
upload: data\weather\nyc_weather.csv to s3://transportation-project-weather/nyc_weather.csv

D:\public-transport-demand>
```



- transportation-project-processed → stores hourly + joined parquet outputs.

- transportation-project-scripts → stores Glue ETL scripts.

3. CLI Setup:

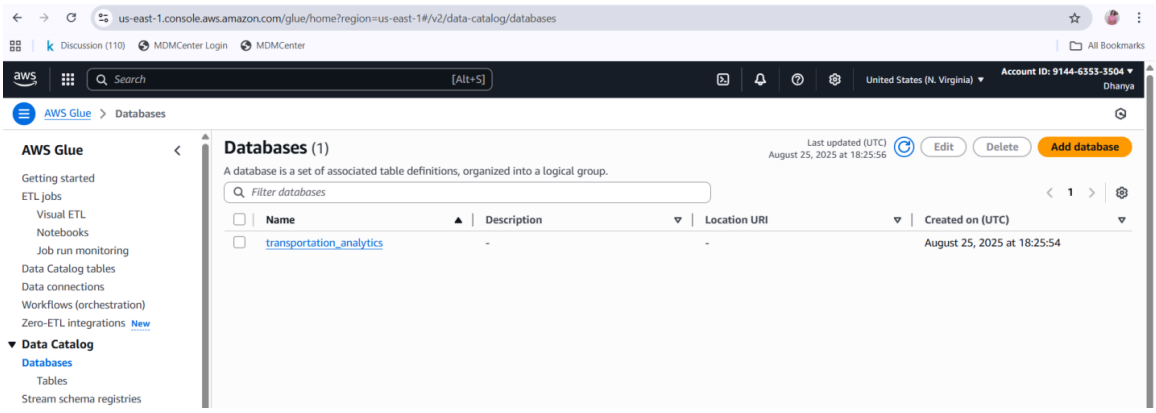
- aws configure → set access keys and region.

- aws s3 mb → created buckets.

AWS Glue Setup

Glue Database:

- Created database: transport_analytics.



Glue Jobs:

1. taxi-transform-job (spark_transform.py):

- Reads raw taxi CSV from S3.

- Extracts date, hour, dayofweek.

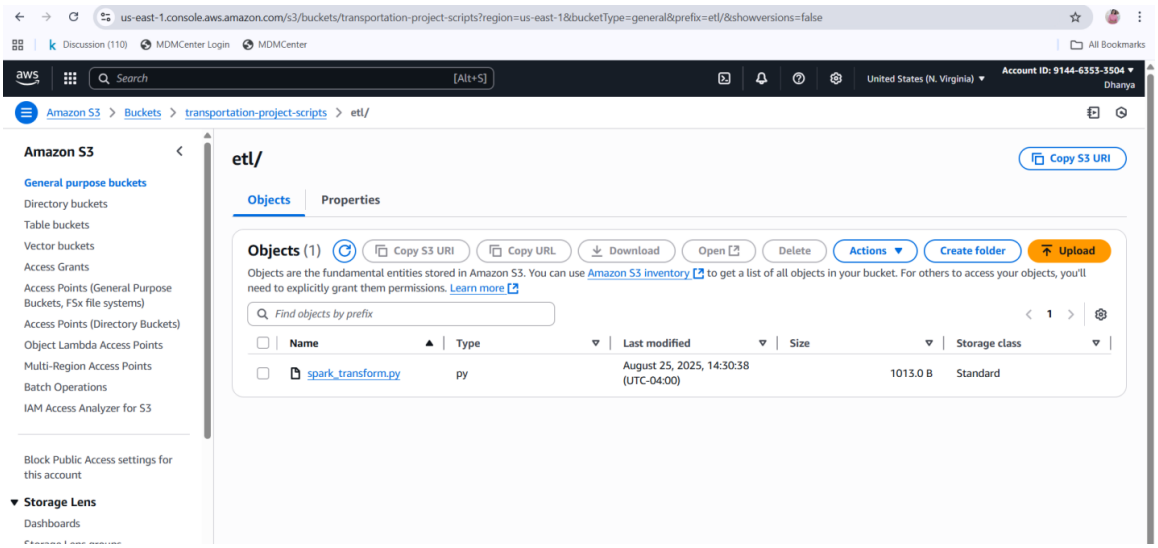
- Aggregates: avg fare, avg distance, trip_count.

- Stores parquet output in processed bucket.

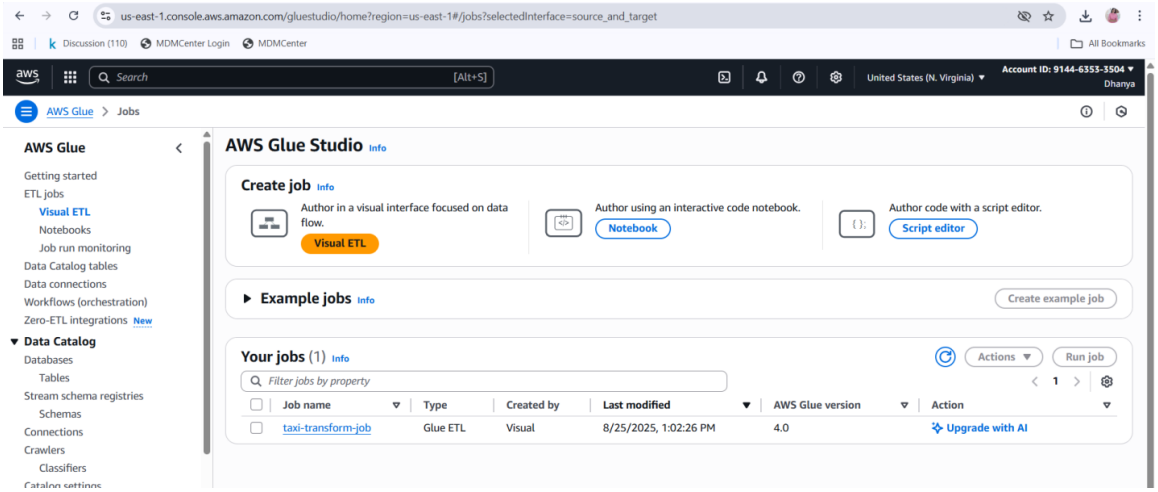
```
etl > spark_transform.py > ...
1  from awsglue.context import GlueContext
2  from pyspark.context import SparkContext
3  from pyspark.sql.functions import hour, dayofweek, to_date
4
5  glueContext = GlueContext(SparkContext.getOrCreate())
6  spark = glueContext.spark_session
7
8  input_path = "s3://transportation-project-raw/taxi/raw/"
9  output_path = "s3://transportation-project-processed/taxi/hourly/"
10
11 df = spark.read.csv(input_path, header=True, inferSchema=True)
12
13 df = df.withColumn("date", to_date("tpep_pickup_datetime")) \
14        .withColumn("hour", hour("tpep_pickup_datetime")) \
15        .withColumn("dayofweek", dayofweek("tpep_pickup_datetime"))
16
17 agg_df = df.groupBy("date", "hour", "dayofweek") \
18            .agg({"fare_amount": "avg", "trip_distance": "avg", "*": "count"}) \
19            .withColumnRenamed("count(1)", "trip_count") \
20            .withColumnRenamed("avg(fare_amount)", "avg_fare") \
21            .withColumnRenamed("avg(trip_distance)", "avg_distance")
22
23 agg_df.write.mode("overwrite").parquet(output_path)
24
```

The file is uploaded to s3://transportation/etl/

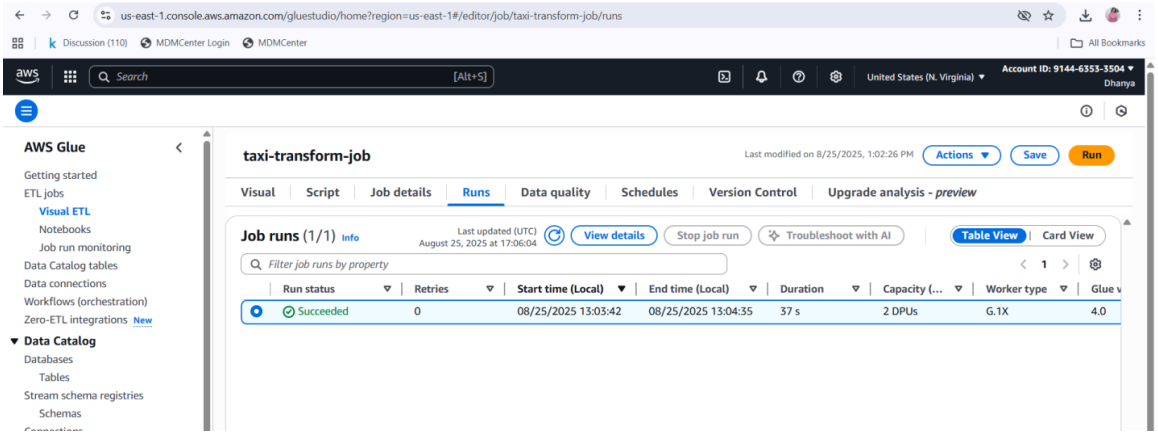
```
D:\public-transport-demand>aws s3 cp etl/spark_transform.py s3://transportation-project-scripts/etl/
upload: etl\spark_transform.py to s3://transportation-project-scripts/etl/spark_transform.py

D:\public-transport-demand>
```



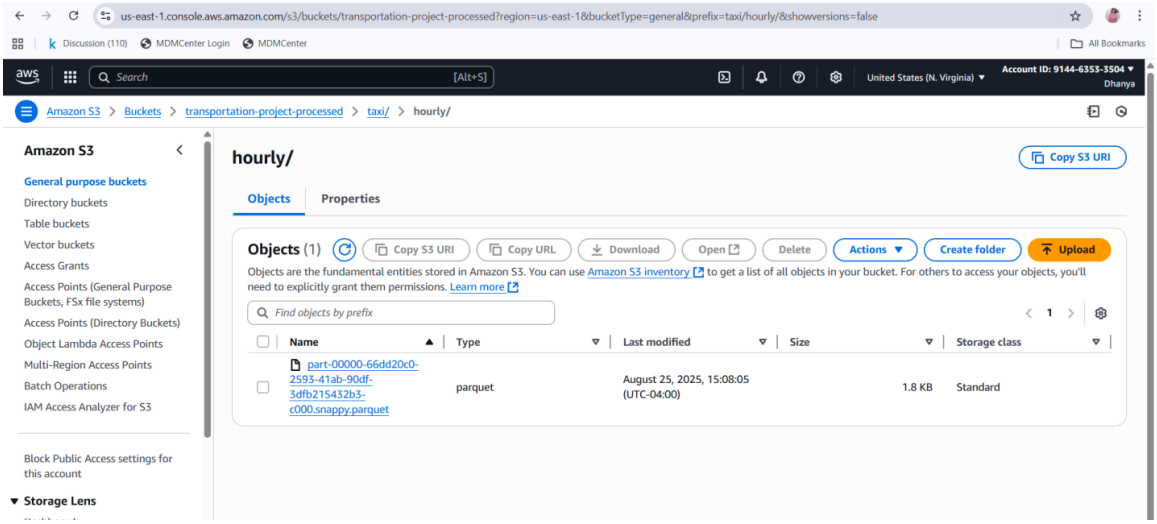
Create taxi-transform-job



Run job taxi-transform-job



File is creates in s3 after running the job



2. taxi-weather-join-job (spark_join_weather.py):

- Reads hourly taxi parquet + weather CSV.

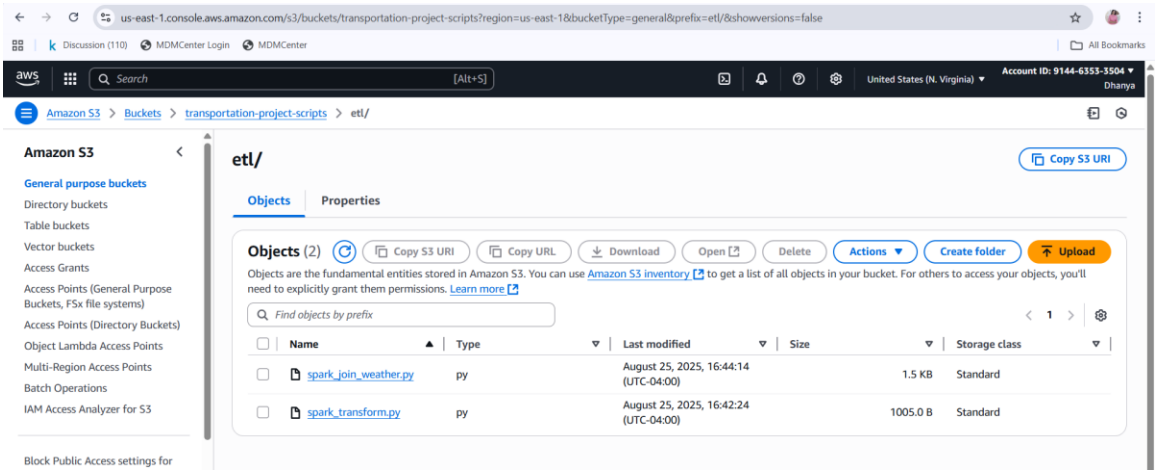
- Joins on date + hour.

- Outputs enriched parquet to S3.

```python
etl > spark_join_weather.py > ...
from awsglue.context import GlueContext
from pyspark.context import SparkContext
from pyspark.sql.functions import to_date, hour
glueContext = GlueContext(SparkContext.getOrCreate())
spark = glueContext.spark_session
# Paths
taxi_path = "s3://transportation-project-processed/taxi/hourly/"
weather_path = "s3://transportation-project-weather/nyc_weather.csv"
output_path = "s3://transportation-project-processed/taxi/joined/"
# Load taxi parquet
taxi_df = spark.read.parquet(taxi_path)
# Explicitly select only needed taxi columns
taxi_df = taxi_df.select("date", "hour", "dayofweek", "trip_count", "avg_fare", "avg_distance")
# Load weather CSV
weather_df = spark.read.csv(weather_path, header=True, inferSchema=True)
# Add derived fields
weather_df = weather_df.withColumn("weather_date", to_date("datetime")) \
                       .withColumn("weather_hour", hour("datetime"))
# Select only required weather columns (rename for clarity)
weather_df = weather_df.select(
    "weather_date",
    "weather_hour",
    "temperature",
    "humidity",
    "pressure",
    "wind_speed"
)
# Perform join on both date and hour
joined_df = taxi_df.join(
    weather_df,
    (taxi_df.date == weather_df.weather_date) & (taxi_df.hour == weather_df.weather_hour),
    how="inner"
)
# Drop duplicate join keys from weather
joined_df = joined_df.drop("weather_date", "weather_hour")
# Write result safely
joined_df.write.mode("overwrite").parquet(output_path)
```

Upload to scripts bucket:

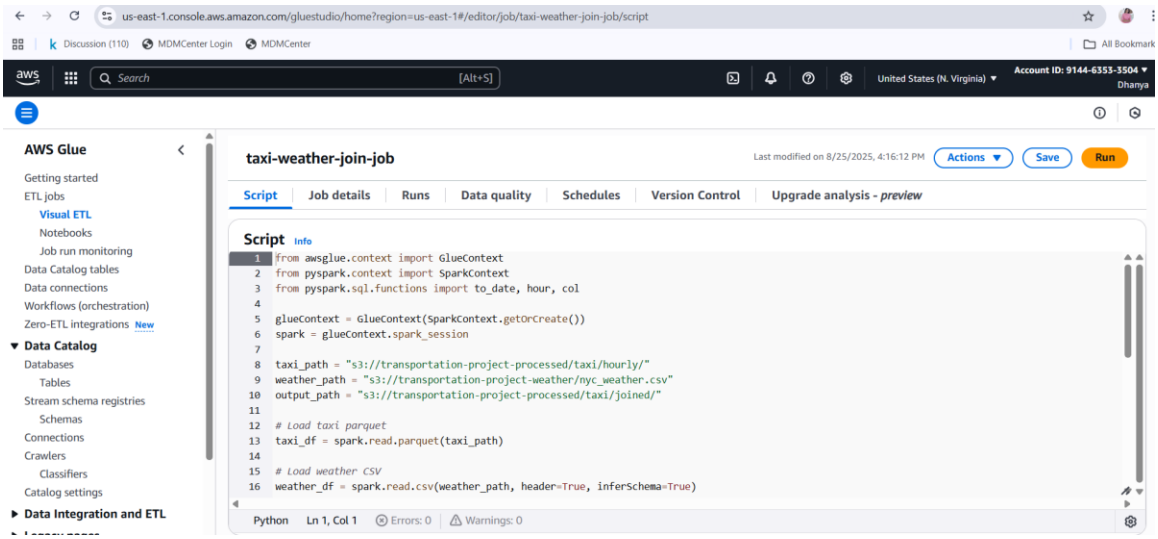
```
D:\public-transport-demand>aws s3 cp etl/spark_join_weather.py s3://transportation-project-scripts/etl/
upload: etl\spark_join_weather.py to s3://transportation-project-scripts/etl/spark_join_weather.py

D:\public-transport-demand>
```



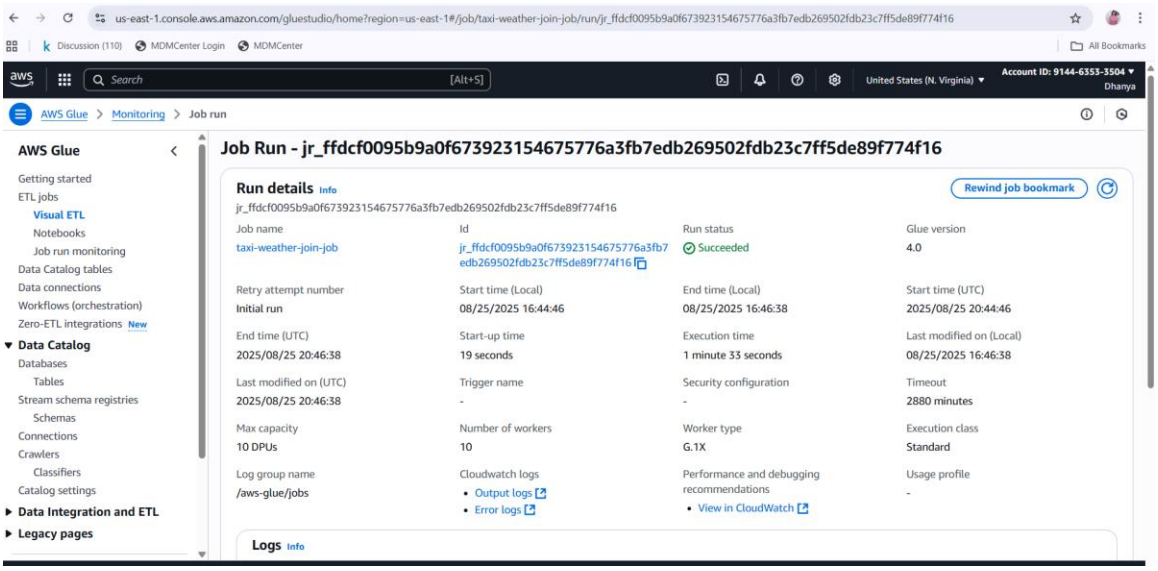
Create job

aws glue create-job --name taxi-weather-join-job --role AWSGlueServiceRole-Transport --command "Name=glueetl,ScriptLocation=s3://transportation-project-scripts/etl/spark_join_weather.py,PythonVersion=3" --region us-east-1

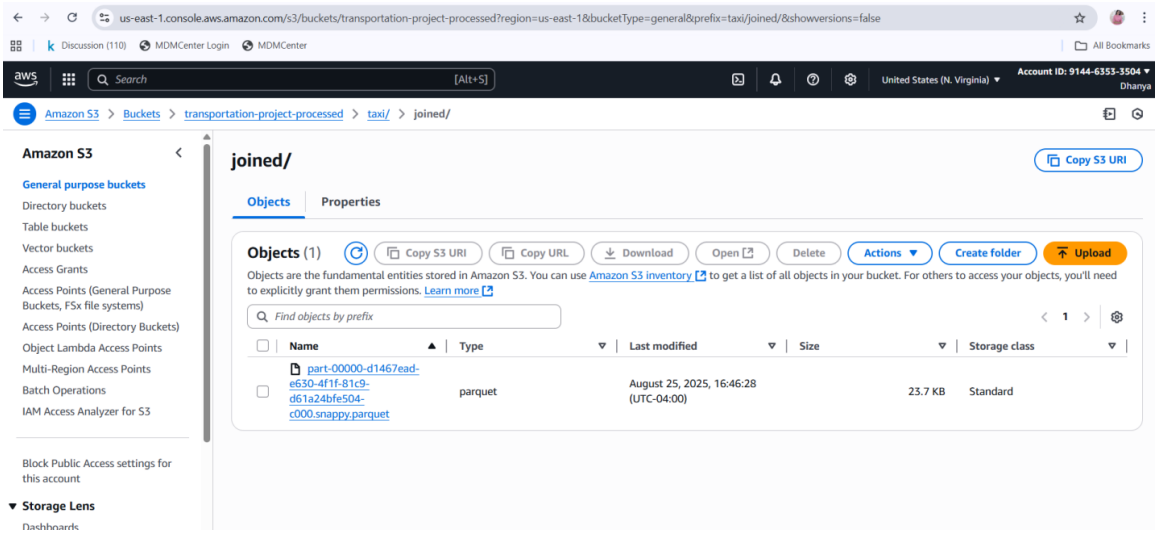
Create job in glue



Run the job



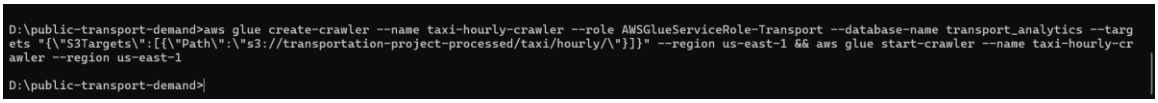
File created successfully in s3 after job run

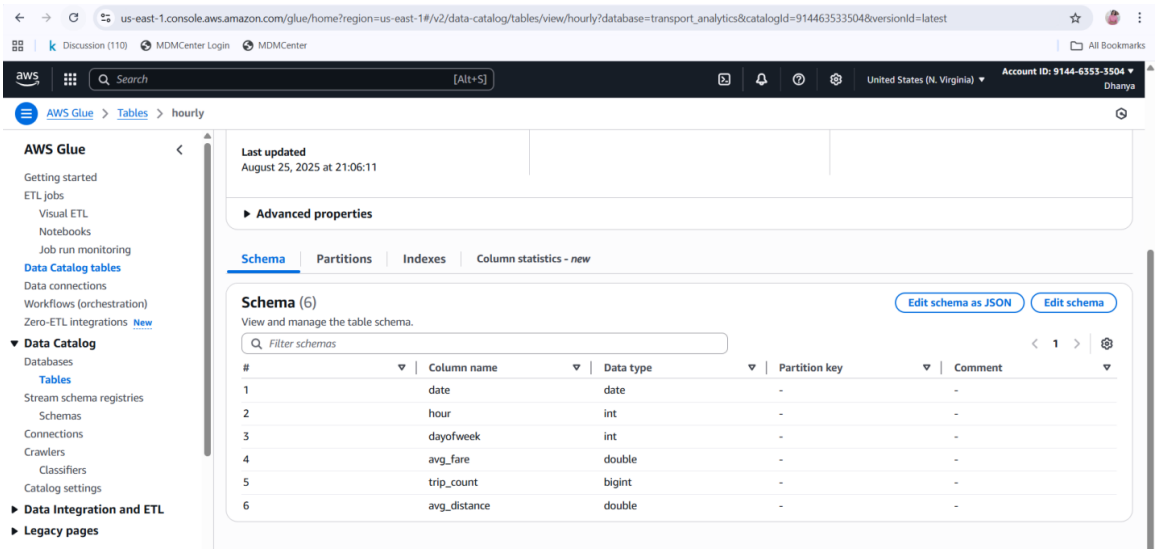


Glue Crawlers:

- taxi-hourly-crawler → points to hourly parquet output.

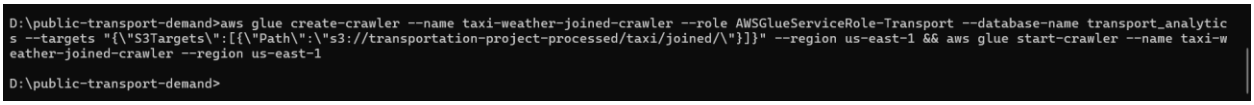
aws glue create-crawler --name taxi-hourly-crawler --role AWSGlueServiceRole-Transport --database-name transport_analytics --targets "{\"S3Targets\":[{\"Path\":\"s3://transportation-project-processed/taxi/hourly/\"}]}" --region us-east-1 && aws glue start-crawler --name taxi-hourly-crawler --region us-east-1

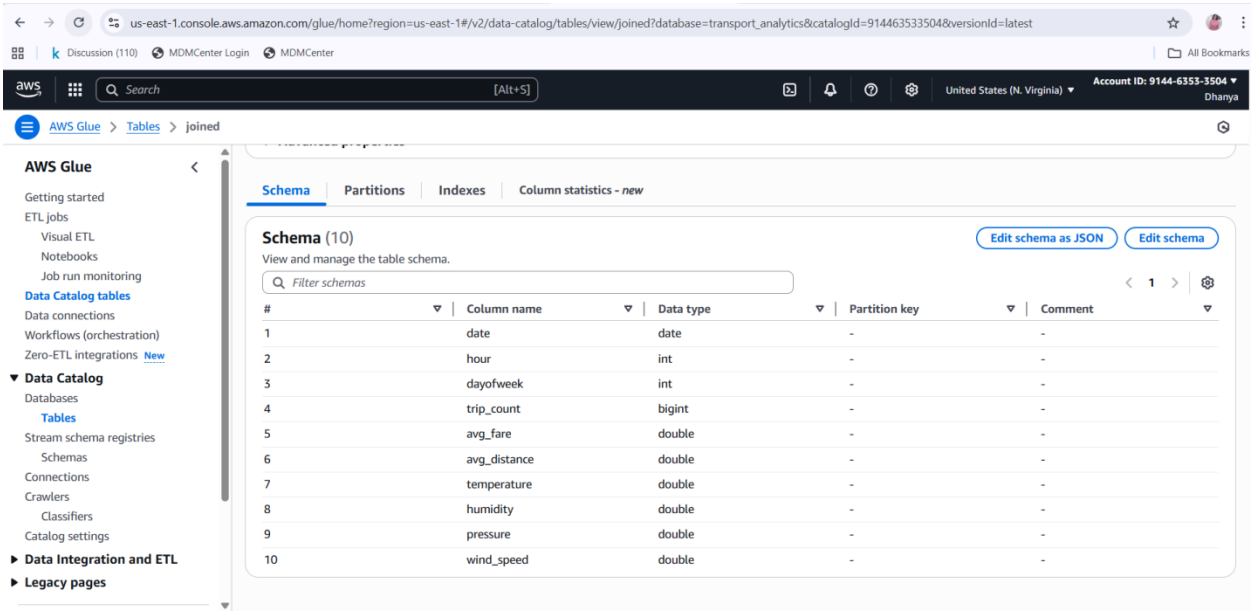




- taxi-weather-joined-crawler → points to joined parquet output.

aws glue create-crawler --name taxi-weather-joined-crawler --role AWSGlueServiceRole-Transport --database-name transport_analytics --targets "{\"S3Targets\":[{\"Path\":\"s3://transportation-project-processed/taxi/joined/\"}]}" --region us-east-1 && aws glue start-crawler --name taxi-weather-joined-crawler --region us-east-1





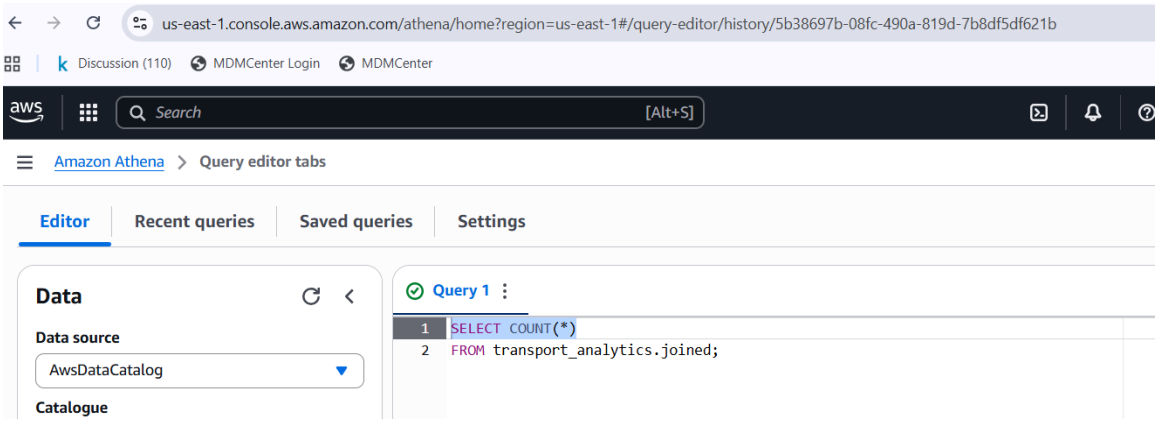
All jobs validated with Spark logs, outputs verified in S3.

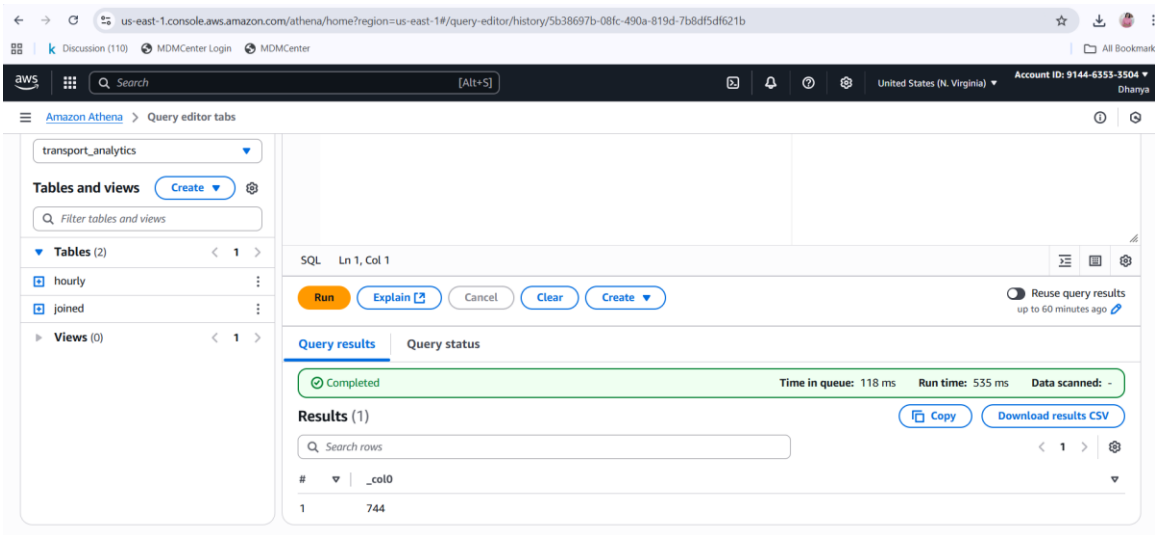
Athena Queries

Key Queries:

1. Validate joined dataset:

SELECT COUNT(*) FROM transport_analytics.joined;





2. Demand by Hour of Day:

SELECT hour, SUM(trip_count) FROM transport_analytics.taxi_weather_joined GROUP BY

hour;

Trips vs Humidity Bands

Query 1 Query 2

```sql
SELECT
  CASE
    WHEN humidity < 40 THEN 'Low'
    WHEN humidity BETWEEN 40 AND 70 THEN 'Medium'
    ELSE 'High'
  END AS humidity_band,
  AVG(trip_count) AS avg_trips
FROM transport_analytics.joined
GROUP BY
  CASE
    WHEN humidity < 40 THEN 'Low'
    WHEN humidity BETWEEN 40 AND 70 THEN 'Medium'
    ELSE 'High'
  END
ORDER BY avg_trips DESC;
```

Query results | Query status

Completed | Time in queue: 80 ms | Run time: 549 ms | Data scanned: 3.98 KB

Results (3) | Copy | Download results CSV

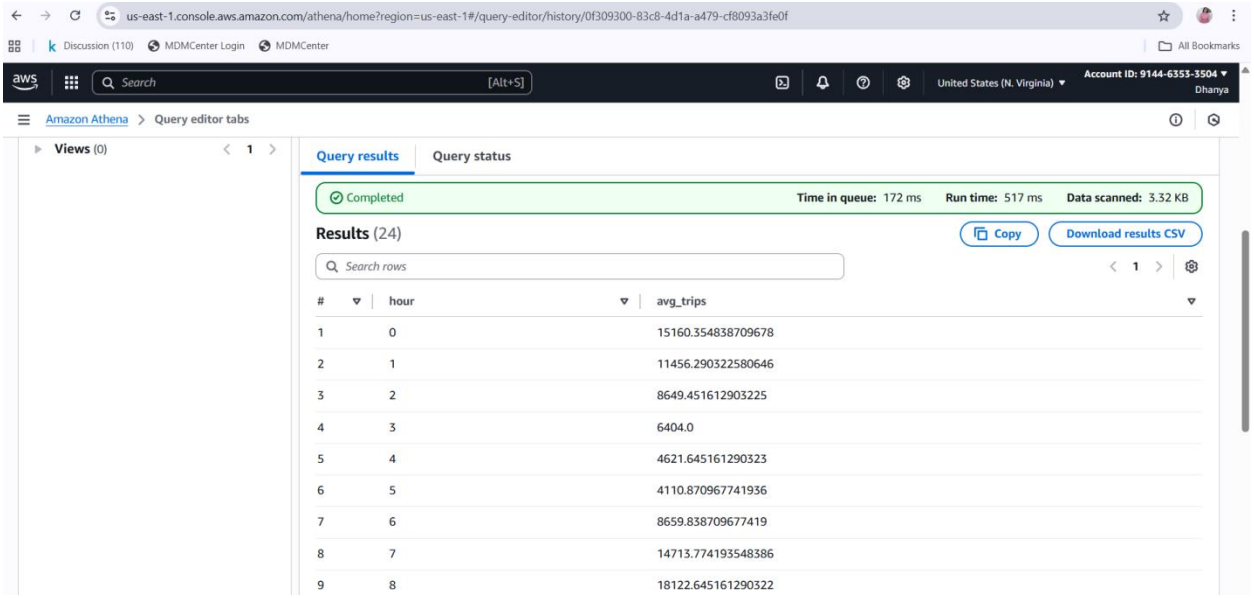
Search rows | 1

#	humidity_band	avg_trips
1	Low	17975.333333333332
2	Medium	17870.37623762376
3	High	16855.74025974026

3. Trips per Hour of Day

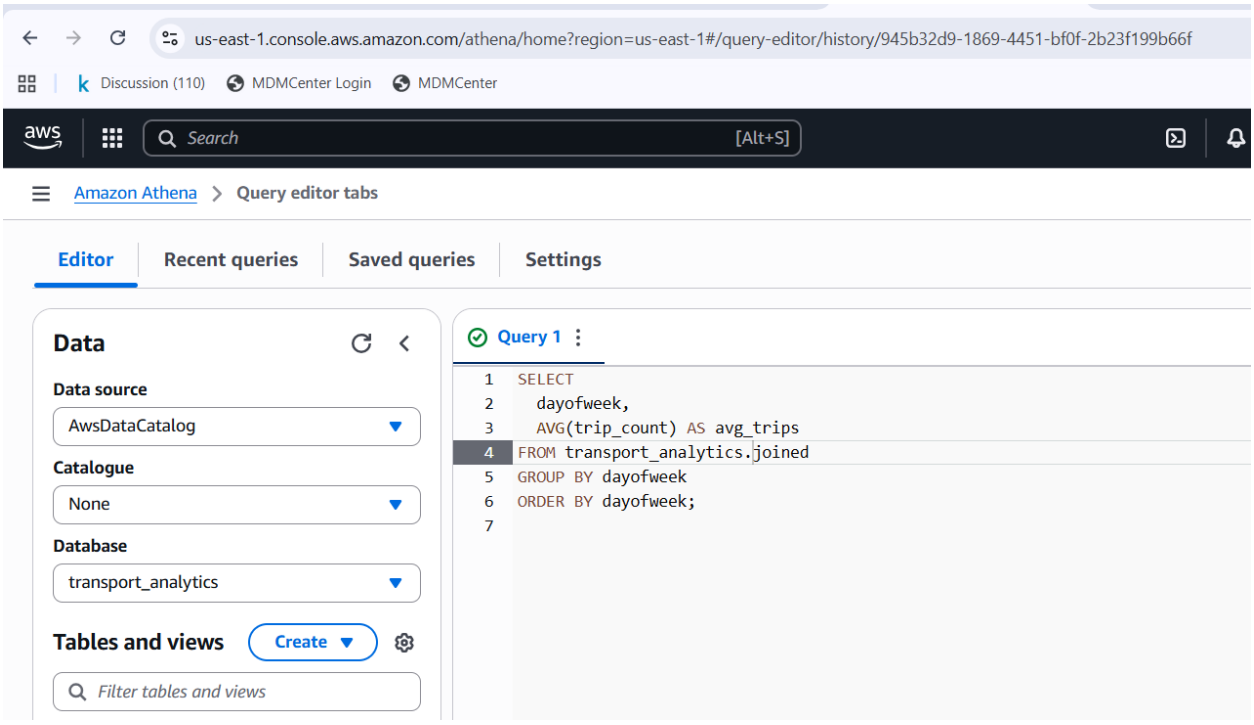
Query 1

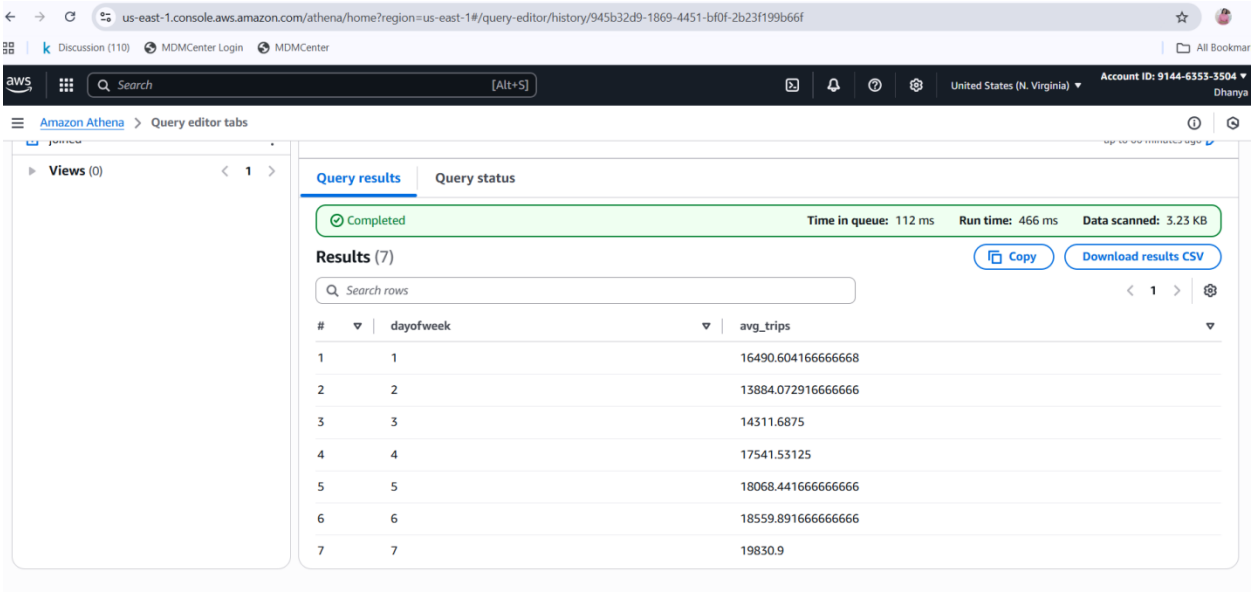
```
1  SELECT
2    hour,
3    AVG(trip_count) AS avg_trips
4  FROM transport_analytics.joined
5  GROUP BY hour
6  ORDER BY hour;
```



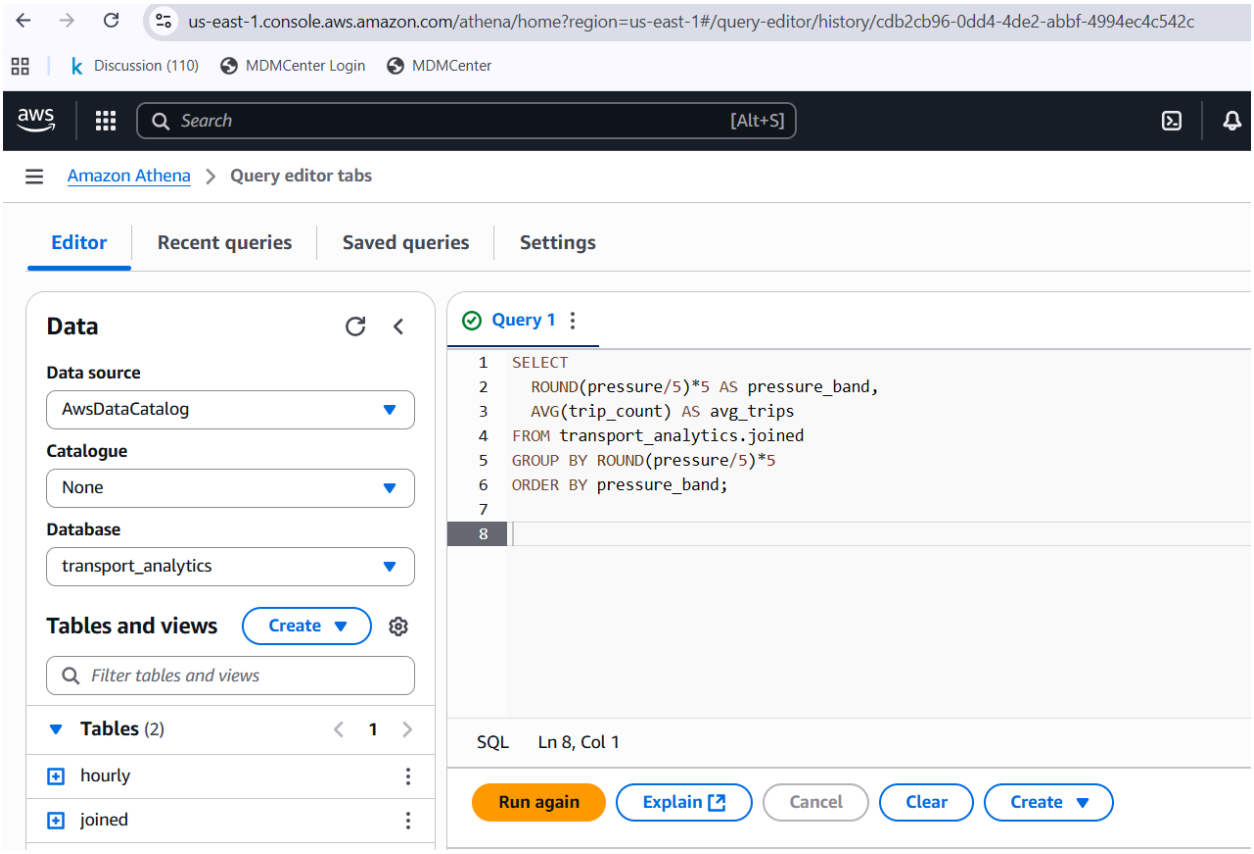
#	hour	avg_trips
1	0	15160.354838709678
2	1	11456.290322580646
3	2	8649.451612903225
4	3	6404.0
5	4	4621.645161290323
6	5	4110.870967741936
7	6	8659.838709677419
8	7	14713.774193548386
9	8	18122.645161290322

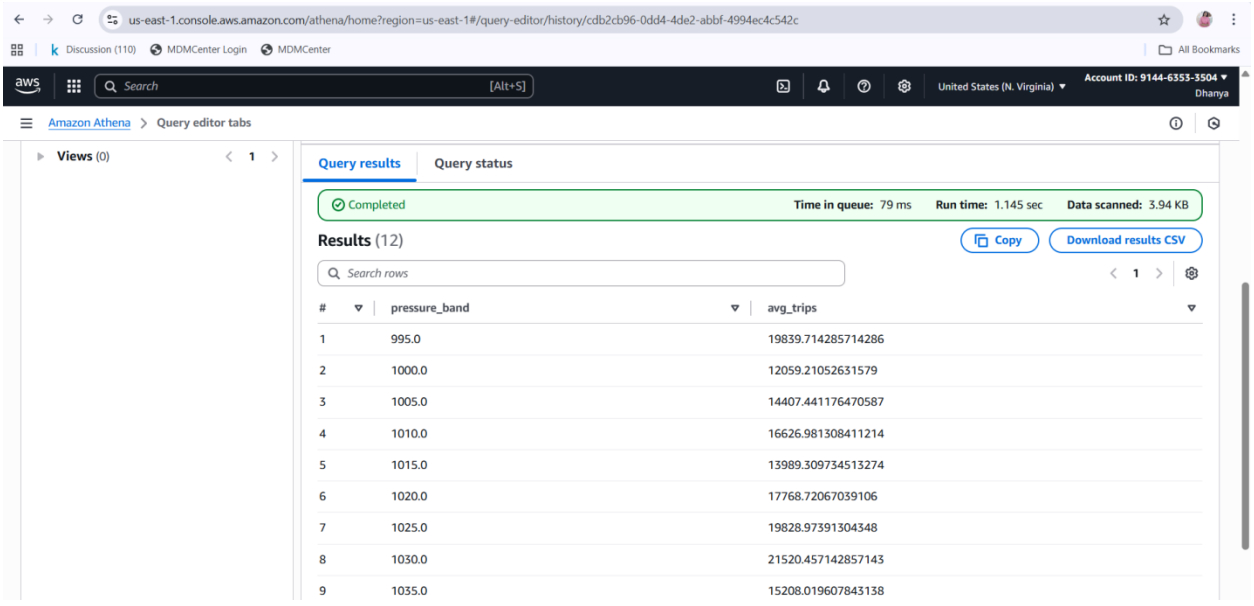
4. Day-of-Week Demand Patterns



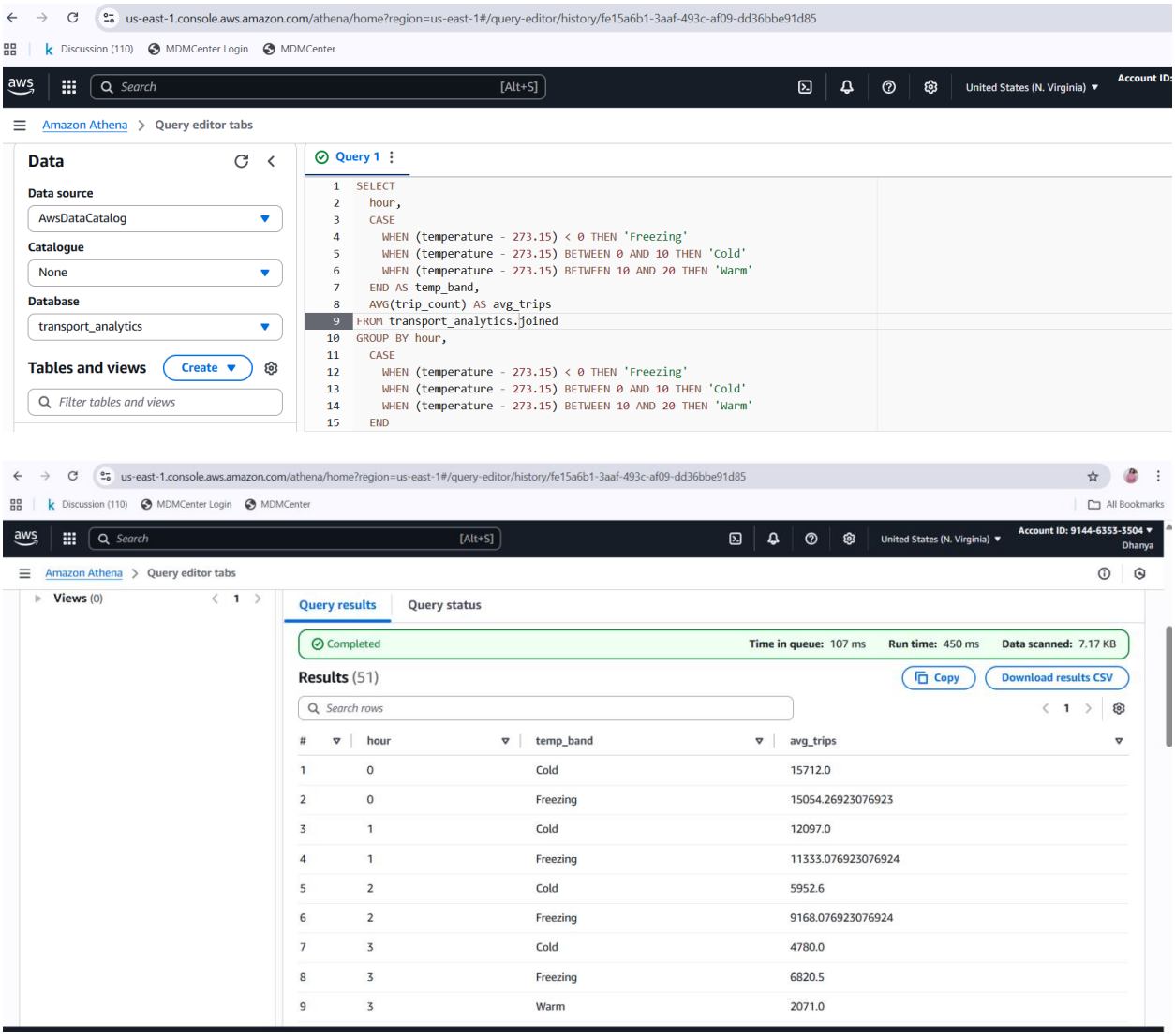


5. Demand vs Pressure

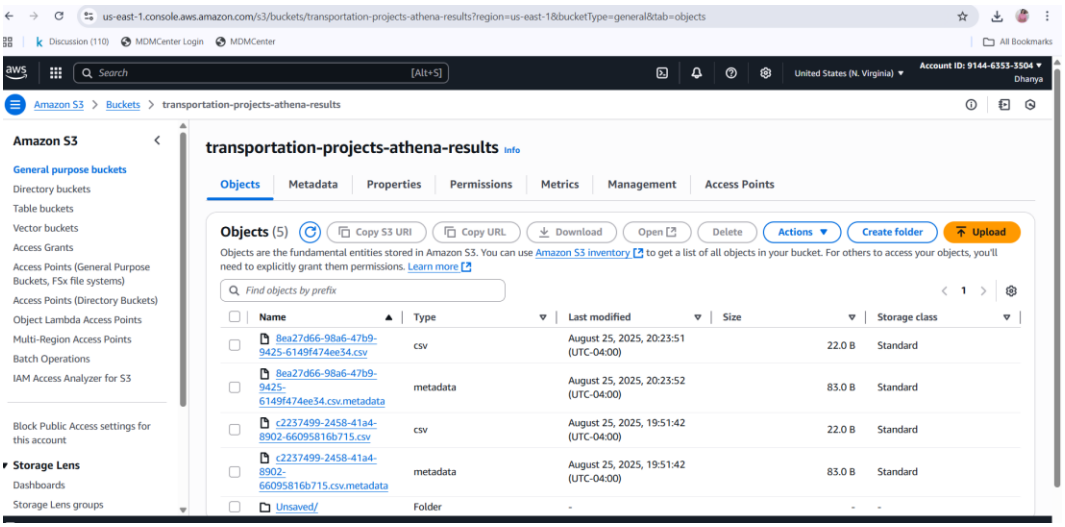




6. Combined Effect: Hour + Temperature Band



Each query validated in Athena console, results stored in athena-results bucket.



QuickSight Dashboards

Steps:

1. Enabled QuickSight with Athena integration.

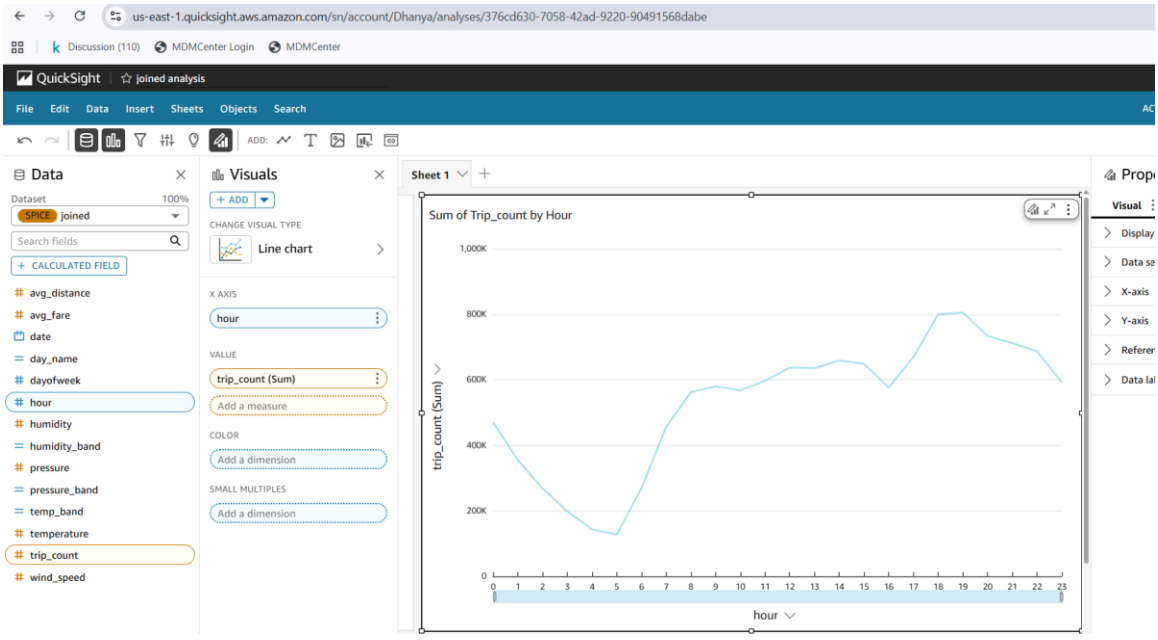
2. Granted QuickSight access to S3 + Glue Data Catalog.

3. Created dataset from Athena table taxi_weather_joined.

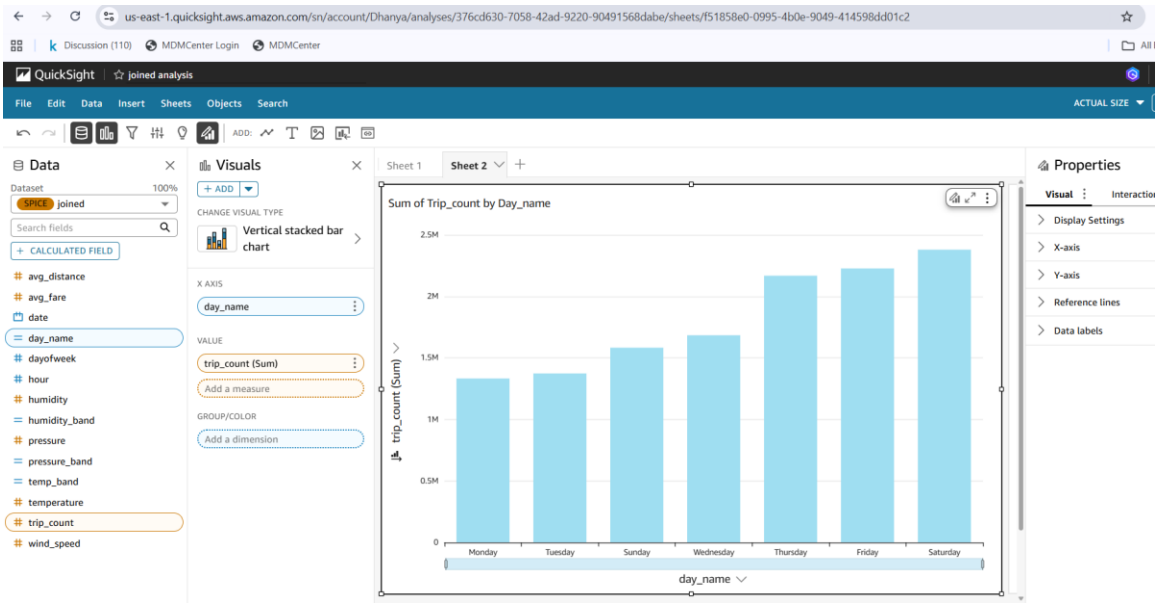
4. Imported into SPICE for performance.

Dashboards Created:

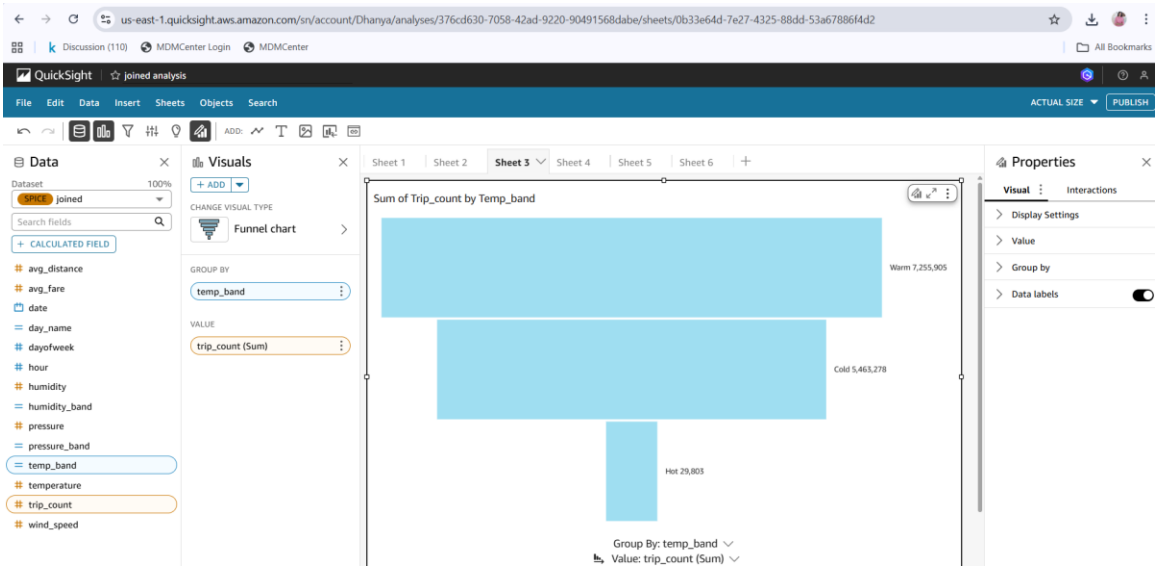
1. Trips by Hour (Line Chart).



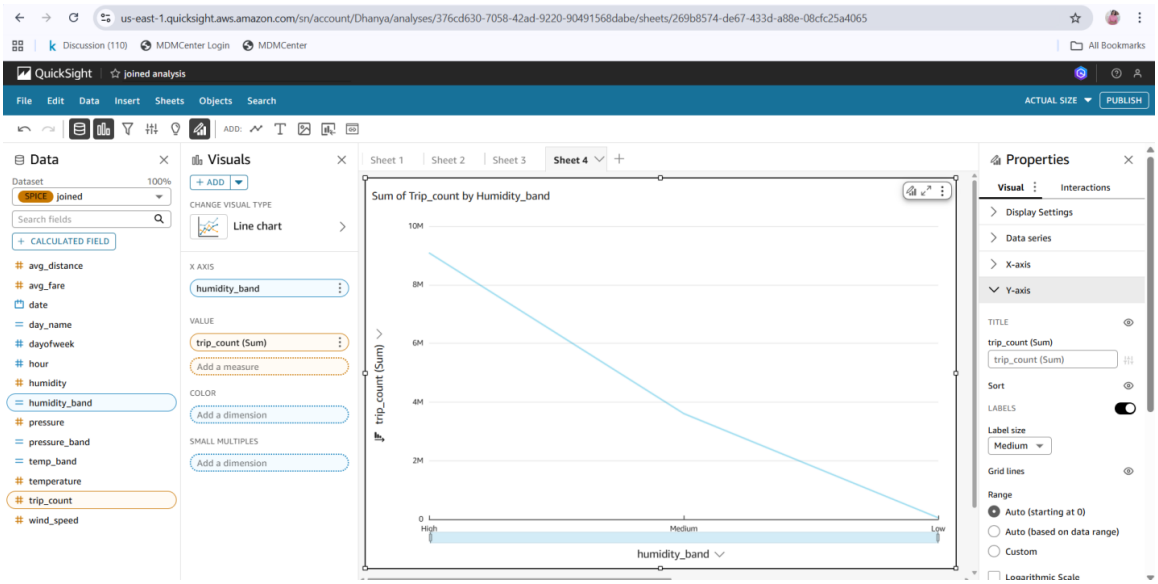
2. Trips by Day of Week (Bar).



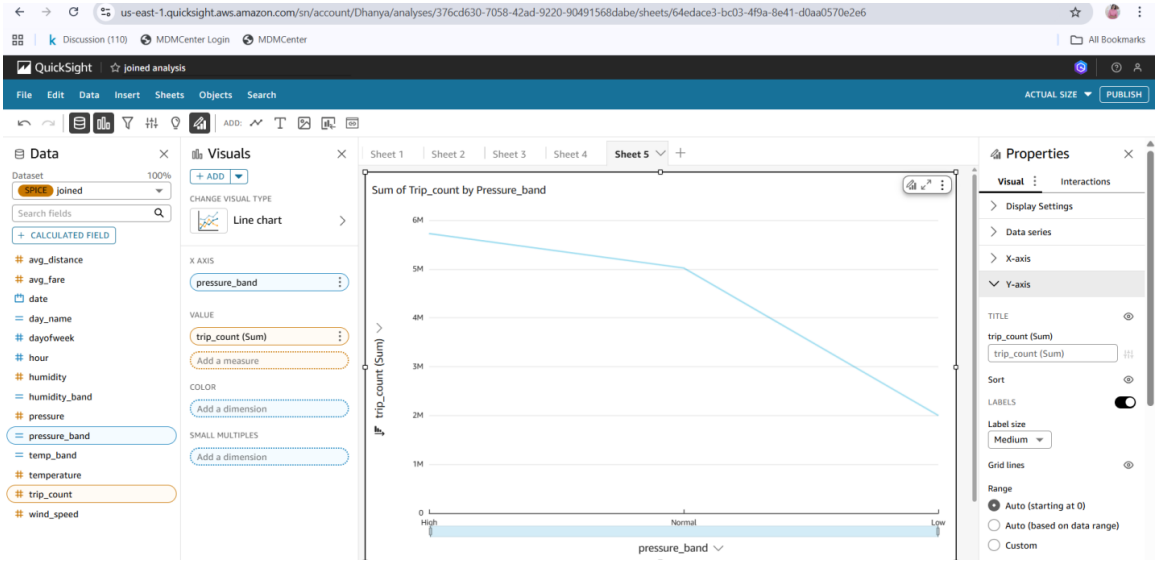
3. Trips vs Temperature Band (Funnel/Bar).



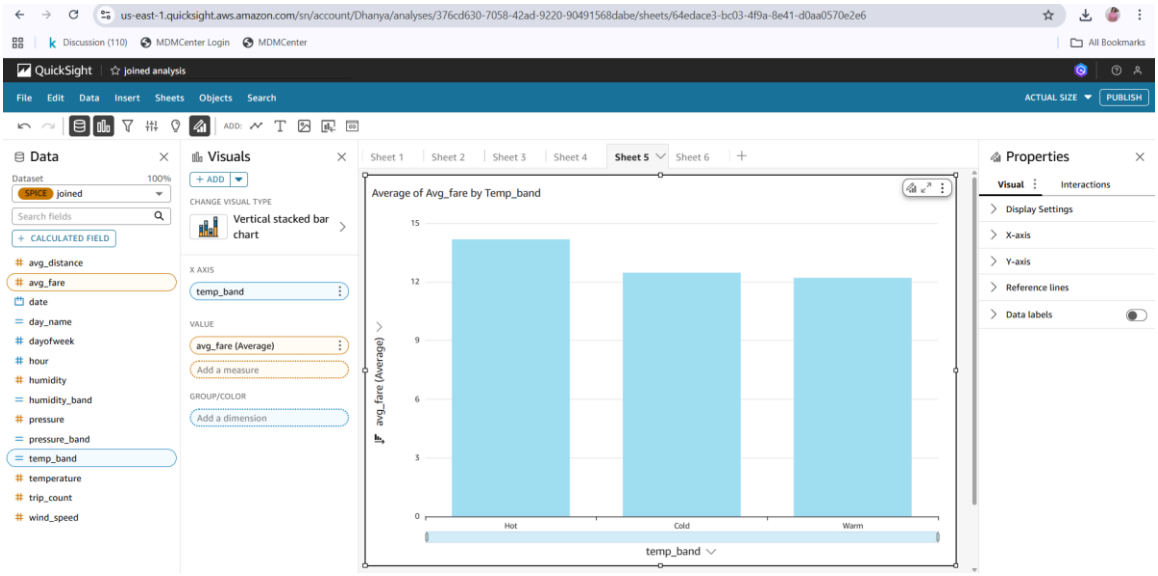
4. Trips vs Humidity (Line).



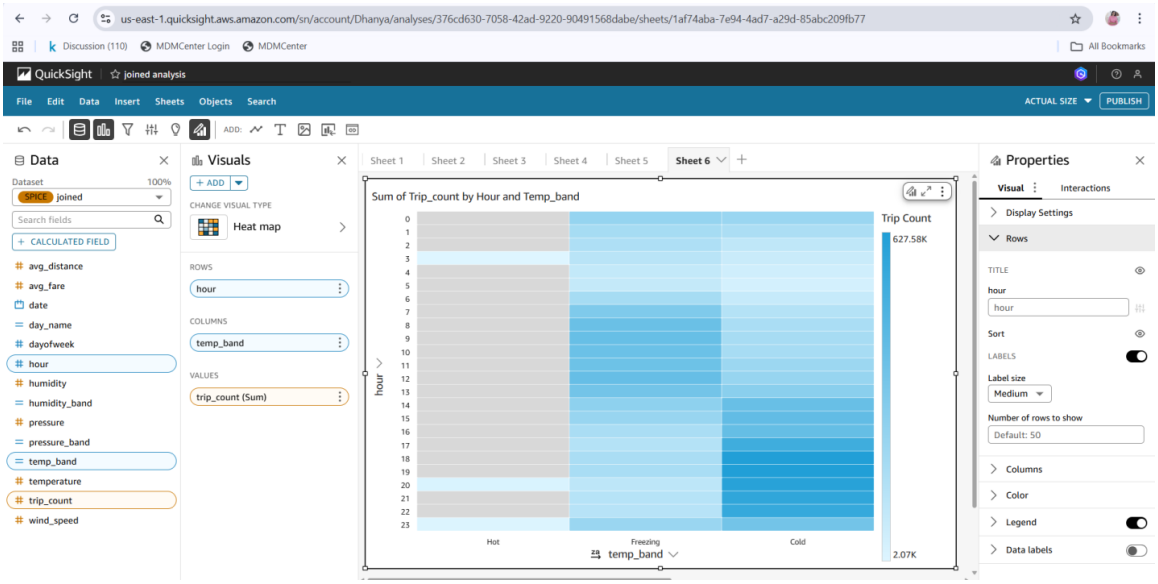
5. Trips vs Pressure (Line).



6. Avg Fare vs Temperature (Bar).



7. Heatmap: Hour × Temperature Band.



Insights:

- Trips by Hour: Clear demand peaks at 18–20 hrs (evening rush), with lowest demand around 4–6 AM.
- Trips by Day of Week:Friday and Saturday show the highest trip volumes, while Monday/Tuesday are lowest.
- Trips vs Temperature Band: Most demand falls in the “Warm” band, with significantly fewer trips in “Hot” or “Cold.”
- Trips vs Humidity/Pressure:Higher humidity and lower pressure correlate with a noticeable drop in trip counts.
- Heatmap (Hour × Temp Band):Evening peaks remain strong across all temperature bands, though “Warm” conditions show the highest demand concentration.

Airflow Orchestration

DAG transport_demand_pipeline:

Tasks:

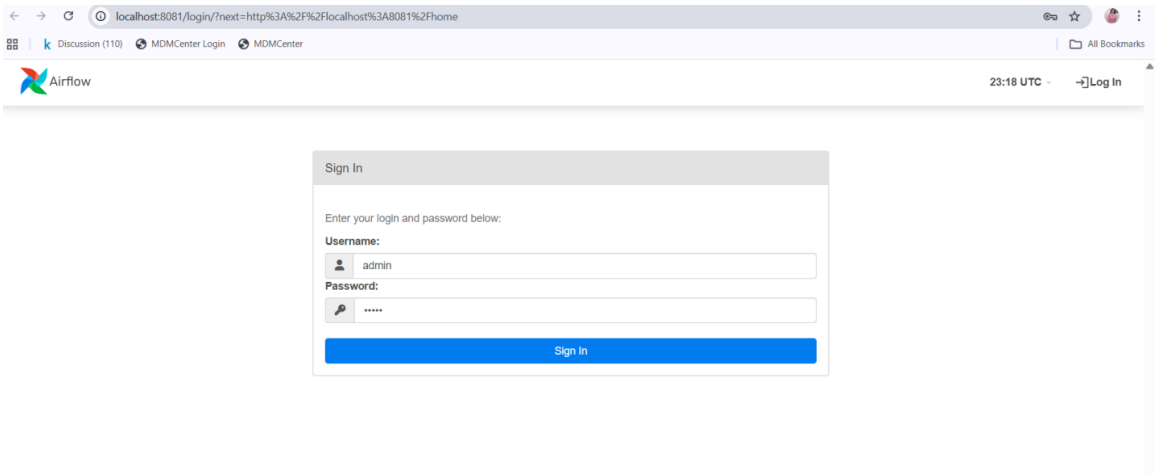
1. Ingest raw taxi/weather CSVs into S3.

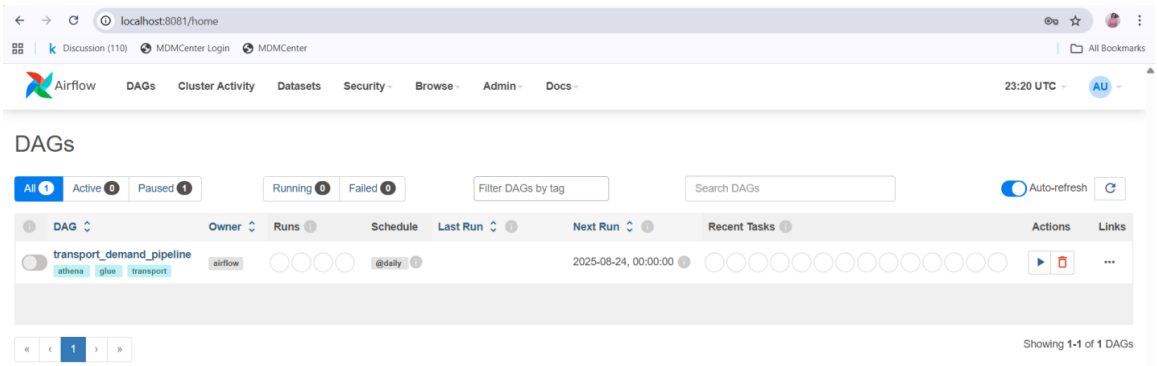
2. Run Glue Job (taxi-transform-job).

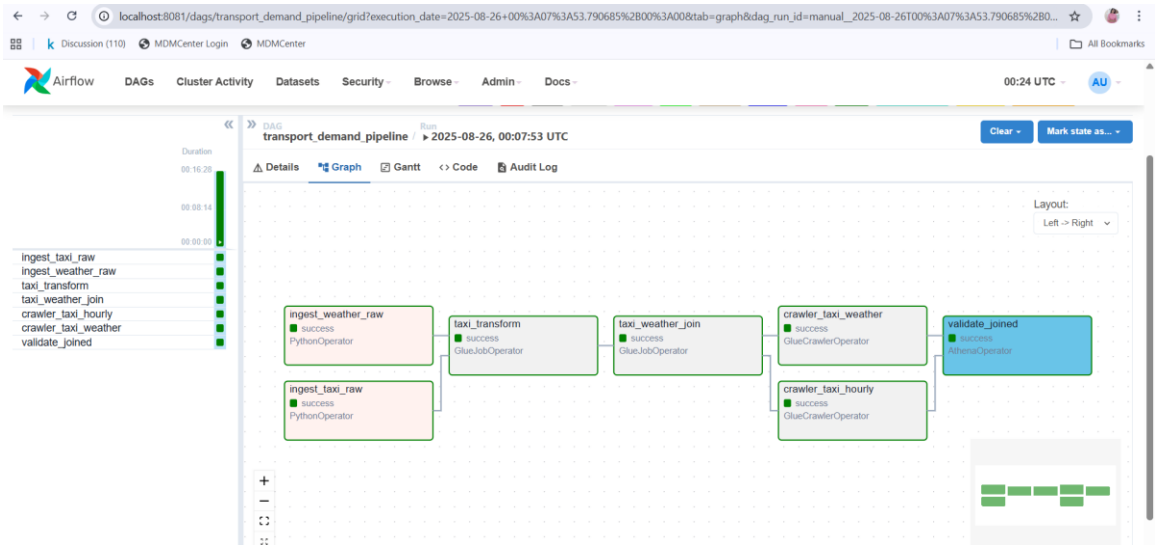
3. Run Glue Job (taxi-weather-join-job).

4. Run Crawlers (taxi-hourly, taxi-weather-joined).

5. Run Athena validation query.







Airflow UI provided DAG Graph, Gantt charts, and detailed logs.

Challenges:

- AWS credentials injection into Docker containers (resolved using ~/.aws mount).

- Duplicate scheduled + manual runs (resolved by disabling catchup and schedule_interval=None).

Results & Analysis

Findings:

- The analysis confirms that demand aggregation is strongly time-driven, with evenings and weekends being critical for capacity planning.

- Weather acts as a secondary but important factor — demand falls during extreme temperature, humidity, or pressure conditions.

- Fare patterns indicate resilience: even when trips decline in bad weather, average fares rise, suggesting longer routes or surge-like effects.

- The combined results highlight that transport operators should align fleet allocation with both predictable cycles (time/day) and unpredictable conditions (weather).

- This validates the project goal: integrating taxi + weather data provides more accurate demand forecasting than relying on trip data alone.

Business Value:

- Taxi fleets can pre-position vehicles for peak hours.

- City planners can adjust transport supply during adverse weather.

- Ride-hailing companies can plan surge pricing better.

Technical Results:

- Glue successfully processed millions of rows efficiently.

- Athena queries executed within seconds thanks to parquet optimization.

- QuickSight dashboards provided interactive insights.

Challenges & Troubleshooting

1. Duplicate Columns Error:

- During join job, duplicate 'hour'/'date' fields caused errors. Fixed by renaming.

2. Airflow:

- "Unable to locate credentials" error fixed by mounting ~/.aws into Docker.

3. Docker/WSL:

- Issues with ports and init resolved by running `airflow db migrate`.

4. QuickSight:

- Temp was in Kelvin; adjusted calculation to Celsius bands.

Conclusion

The project successfully demonstrates an end-to-end cloud ETL + analytics pipeline.

It integrates raw and external datasets, transforms them, validates results, and generates business dashboards.

The architecture is modular, scalable, and can be extended to other data sources.

Future Scope:

- Add real-time ingestion with Kinesis + Lambda.

- ML-based demand prediction models.

- Integration with city traffic/event datasets.

Cost Optimization Strategies

While building the Public Transport Demand Aggregation Pipeline, several opportunities for cost savings were identified. These optimizations ensure that the solution can remain efficient and scalable in a production environment:

1. Amazon S3

- Use lifecycle rules to automatically transition old raw data to cheaper storage classes (S3 Standard-IA or Glacier).
- Store processed data in Parquet format to reduce Athena query scan size and cost.

2. AWS Glue

- Enable job bookmarks to prevent reprocessing of already transformed data.
- Choose the smallest worker type (G.1X) sufficient for the dataset to save compute cost.
- Merge transformations into fewer Glue jobs to minimize orchestration overhead.

3. Amazon Athena

- Partition tables by date and hour so queries only scan relevant data.
- Use compressed columnar storage (Parquet) instead of raw CSVs for analytics.

4. Amazon QuickSight

- Use SPICE storage to reduce repeated Athena queries.
- Schedule dataset refresh at daily or weekly frequency rather than every query, balancing cost and performance.

5. Apache Airflow

- Run orchestration locally (via Docker) during development, avoiding managed service costs.
- Schedule Glue and Athena jobs once per day instead of running continuously, reducing execution charges.

GitHub Repository

The complete project codebase, datasets (sample), ETL scripts, Airflow DAGs, and documentation are available on GitHub.

https://github.com/Dhanyamol-Devassy/public-transport-demand-aggregation-aws