

Public Transport Demand Aggregation

Dhanyamol Devassy

Contents

Executive Summary.....	4
Introduction	4
Objective:	4
Architecture Overview.....	5
Architecture Components:.....	5
Data Flow.....	5
Architecture Diagram.....	6
Datasets	7
1. NYC Yellow Taxi Dataset.....	7
2. Weather Dataset	7
Schema Examples	7
Environment Setup.....	7
1. IAM Role.....	7
2. S3 Buckets	8
3. CLI Setup	9
AWS Glue Setup.....	10
Glue Database.....	10
Glue Jobs.....	10
1. taxi-transform-job (spark_transform.py):.....	10
2. taxi-weather-join-job (spark_join_weather.py):	13
Glue Crawlers	16
Athena Queries	18
1. Validate joined dataset	18
2. Demand by Hour of Day	19
3. Trips per Hour of Day.....	20
4. Day-of-Week Demand Patterns	21
5. Demand vs Pressure	22
6. Combined Effect: Hour + Temperature Band	23
QuickSight Dashboards	24
Steps.....	24

Dashboards Created.....	25
1. Trips by Hour (Line Chart).....	25
2. Trips by Day of Week (Bar).....	25
3. Trips vs Temperature Band (Funnel/Bar).....	26
4. Trips vs Humidity (Line).....	26
5. Trips vs Pressure (Line).....	27
6. Avg Fare vs Temperature (Bar).....	27
7. Heatmap: Hour × Temperature Band.....	28
Insights:.....	28
Airflow Orchestration	28
Tasks	29
Challenges	30
Results & Analysis.....	30
Findings.....	30
Business Value	31
Challenges & Troubleshooting.....	31
Conclusion.....	32
Cost Optimization Strategies.....	32
GitHub Repository.....	33

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:

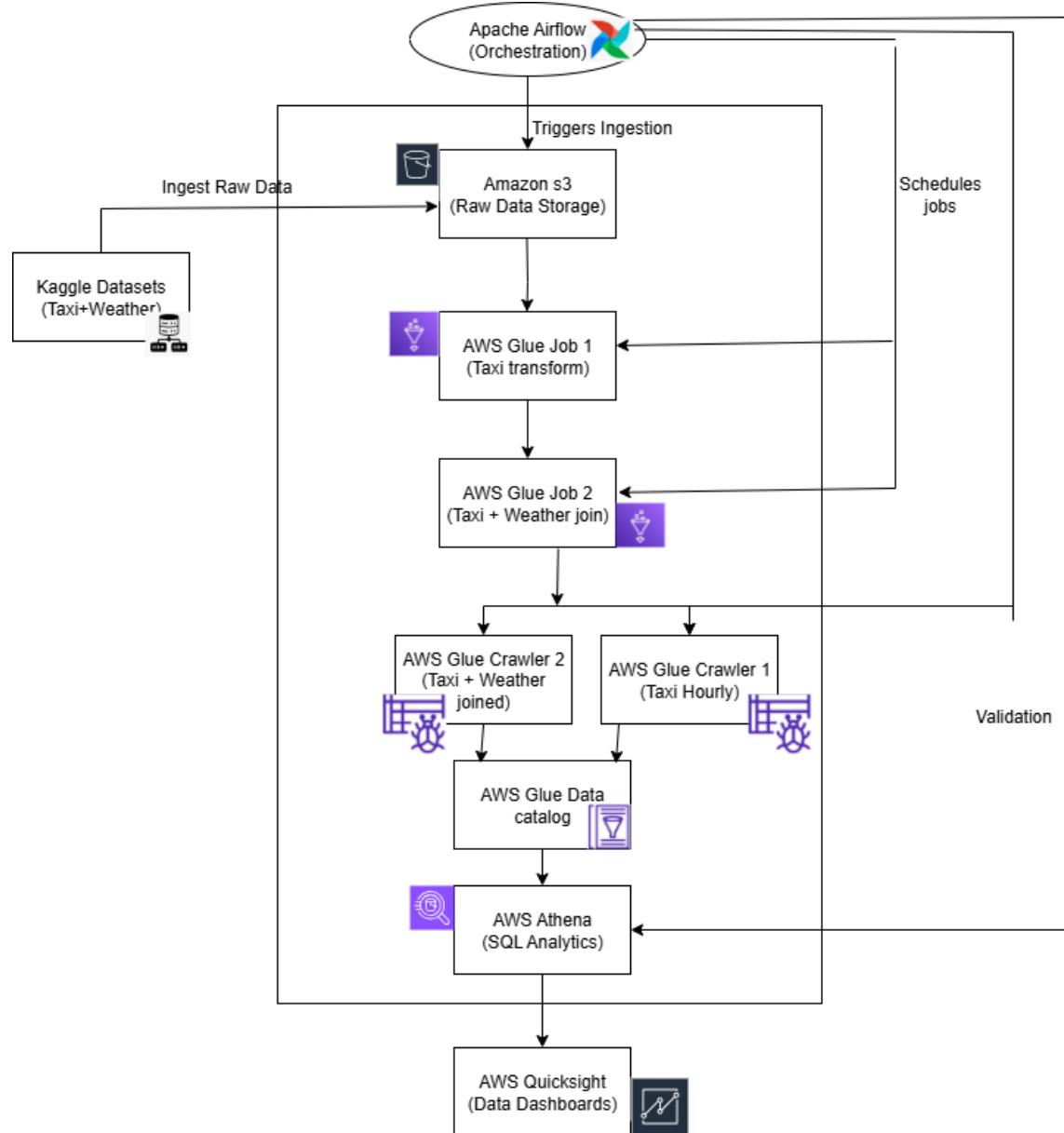
Service	Purpose in Project
Amazon S3	Storage for raw taxi data, weather data, and processed outputs. Acts as a central data lake.
AWS Glue	Serverless ETL for transforming raw taxi trip data and enriching it with weather features.
AWS Glue Crawlers	Automatically detect schema and create tables in the 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.

AWSGlueServiceRole-Transport Info

Allows Glue to call AWS services on your behalf.

Summary

ARN
arn:aws:iam::914463533504:role/AWSGlueServiceRole-Transport

Maximum session duration
1 hour

Permissions Trust relationships Tags Last Accessed Revoke sessions

Permissions policies (3) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonAthenaFullAccess	AWS managed	1

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

General purpose buckets (4) Info

Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
transportation-project-processed	US East (N. Virginia) us-east-1	August 25, 2025, 14:17:36 (UTC-04:00)
transportation-project-raw	US East (N. Virginia) us-east-1	August 25, 2025, 14:17:18 (UTC-04:00)
transportation-project-scripts	US East (N. Virginia) us-east-1	August 25, 2025, 14:17:50 (UTC-04:00)
transportation-project-weather	US East (N. Virginia) us-east-1	August 25, 2025, 14:17:44 (UTC-04:00)

Account snapshot Info

Updated daily
Storage Lens provides visibility into storage usage and activity trends.

External access summary - NEW Info

Updated daily
External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

- 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
```

raw/

Objects **Properties**

Objects (1)

Name	Type	Last modified	Size	Storage class
yellow_tripdata_2015-01.csv	csv	August 25, 2025, 14:18:32 (UTC-04:00)	1.8 GB	Standard

- 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-weather [Info](#)

Objects **Metadata** **Properties** **Permissions** **Metrics** **Management** **Access Points**

Objects (1)

Name	Type	Last modified	Size	Storage class
nyc_weather.csv	csv	August 25, 2025, 14:22:56 (UTC-04:00)	2.5 MB	Standard

- 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`.

The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a navigation sidebar with options like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', and 'Zero-ETL integrations'. Under 'Data Catalog', the 'Databases' section is selected, showing 'Tables' and 'Stream schema registries'. The main content area is titled 'Databases (1)' and contains a table with one row. The table has columns for 'Name', 'Description', 'Location URI', and 'Created on (UTC)'. The single entry is 'transportation_analytics', with a creation date of 'August 25, 2025 at 18:25:54'. There are buttons for 'Edit', 'Delete', and 'Add database' at the top right of the table.

Name	Description	Location URI	Created on (UTC)
transportation_analytics			August 25, 2025 at 18:25:54

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>

```

The screenshot shows the AWS S3 console interface. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/transportation-project-scripts?region=us-east-1&bucketType=general&prefix=etl/8&showversions=false`. The browser title is "Amazon S3". The left sidebar shows "General purpose buckets" with options like "Directory buckets", "Table buckets", "Vector buckets", "Access Grants", "Access Points (General Purpose Buckets, Fsx file systems)", "Access Points (Directory Buckets)", "Object Lambda Access Points", "Multi-Region Access Points", "Batch Operations", and "IAM Access Analyzer for S3". Below this is a "Block Public Access settings for this account" section. The main content area shows the "etl/" folder. The "Objects" tab is selected, displaying a table with one item:

Name	Type	Last modified	Size	Storage class
spark_transform.py	py	August 25, 2025, 14:30:38 (UTC-04:00)	1013.0 B	Standard

Create taxi-transform-job

The screenshot shows the AWS Glue Studio interface. On the left, there's a sidebar with 'AWS Glue' and 'Data Catalog' sections. The main area is titled 'AWS Glue Studio' and shows 'Create job' options: 'Visual ETL' (selected), 'Notebooks', and 'Script editor'. Below this is a 'Example jobs' section with a 'Create example job' button. The central part displays 'Your jobs (1)' with a table showing one job: 'taxi-transform-job' (Glue ETL, Visual, created on 8/25/2025, 1:02:26 PM, version 4.0). There are 'Actions' and 'Run job' buttons above the table.

Run job taxi-transform-job

This screenshot shows the 'taxi-transform-job' details page. The top navigation bar includes 'Actions', 'Save', and a 'Run' button. The main content area is titled 'taxi-transform-job' and shows tabs for 'Visual', 'Script', 'Job details', 'Runs' (selected), 'Data quality', 'Schedules', 'Version Control', and 'Upgrade analysis - preview'. Under the 'Runs' tab, it says 'Job runs (1/1)' last updated on August 25, 2025, at 17:06:04. A table lists one run: 'Succeeded' (0 retries, start time 08/25/2025 13:03:42, end time 08/25/2025 13:04:35, duration 37 s, 2 DPU, G.1X, version 4.0).

File is creates in s3 after running the job

The screenshot shows the AWS S3 console interface. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/transportation-project-processed?region=us-east-1&bucketType=general&prefix=taxi/hourly/&showversions=false`. The top navigation bar includes links for Discussion (110), MDMCenter Login, MDMCenter, and All Bookmarks. The account ID is 9144-6353-5504, and the user is Dhanya. The main content area shows the 'hourly/' folder under the 'taxi/' bucket. The 'Objects' tab is selected, displaying one item: 'part-00000-66dd20c0-2593-41ab-90df-3dfb215432b3-c000.snappy.parquet'. The object details show it is a parquet file, last modified on August 25, 2025, at 15:08:05 (UTC-04:00), with a size of 1.8 KB and a storage class of Standard.

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

- Reads hourly taxi parquet + weather CSV.
- Joins on date + hour.
- Outputs enriched parquet to S3.

```

etl > spark_join_weather.py > ...
1  from awsglue.context import GlueContext
2  from pyspark.context import SparkContext
3  from pyspark.sql.functions import to_date, hour
4  glueContext = GlueContext(sparkContext.getOrCreate())
5  spark = glueContext.spark_session
6  # Paths
7  taxi_path = "s3://transportation-project-processed/taxi/hourly/"
8  weather_path = "s3://transportation-project-weather/nyc_weather.csv"
9  output_path = "s3://transportation-project-processed/taxi/joined/"
10 # Load taxi parquet
11 taxi_df = spark.read.parquet(taxi_path)
12 # Explicitly select only needed taxi columns
13 taxi_df = taxi_df.select("date", "hour", "dayofweek", "trip_count", "avg_fare", "avg_distance")
14 # Load weather CSV
15 weather_df = spark.read.csv(weather_path, header=True, inferSchema=True)
16 # Add derived fields
17 weather_df = weather_df.withColumn("weather_date", to_date("datetime")) \
18 |           .withColumn("weather_hour", hour("datetime"))
19 # Select only required weather columns (rename for clarity)
20 weather_df = weather_df.select(
21     "weather_date",
22     "weather_hour",
23     "temperature",
24     "humidity",
25     "pressure",
26     "wind_speed"
27 )
28 # Perform join on both date and hour
29 joined_df = taxi_df.join(
30     weather_df,
31     (taxi_df.date == weather_df.weather_date) & (taxi_df.hour == weather_df.weather_hour),
32     how="inner"
33 )
34 # Drop duplicate join keys from weather
35 joined_df = joined_df.drop("weather_date", "weather_hour")
36 # Write result safely
37 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>

```

The screenshot shows the AWS S3 console interface. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/transportation-project-scripts?region=us-east-1&bucketType=general&prefix=etl/&showversions=false`. The left sidebar shows 'Amazon S3' and 'General purpose buckets'. The main content area shows the 'etl/' folder under 'Buckets > transportation-project-scripts > etl/'. The 'Objects' tab is selected, displaying two items:

Name	Type	Last modified	Size	Storage class
<code>spark_join_weather.py</code>	py	August 25, 2025, 16:44:14 (UTC-04:00)	1.5 KB	Standard
<code>spark_transform.py</code>	py	August 25, 2025, 16:42:24 (UTC-04:00)	1005.0 B	Standard

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

The screenshot shows the AWS Glue Studio interface. On the left, there's a sidebar with navigation links for AWS Glue, ETL jobs, Visual ETL, Data Catalog, and Data Integration and ETL. The main area is titled 'taxi-weather-join-job' and has tabs for Script, Job details, Runs, Data quality, Schedules, Version Control, and Upgrade analysis - preview. The Script tab is active, showing the following Python code:

```
1 from awsglue.context import GlueContext
2 from pyspark.context import SparkContext
3 from pyspark.sql.functions import to_date, hour, col
4
5 glueContext = GlueContext(SparkContext.getOrCreate())
6 spark = glueContext.spark_session
7
8 taxi_path = "s3://transportation-project-processed/taxi/hourly/"
9 weather_path = "s3://transportation-project-weather/nyc_weather.csv"
10 output_path = "s3://transportation-project-processed/taxi/joined/"
11
12 # Load taxi parquet
13 taxi_df = spark.read.parquet(taxi_path)
14
15 # Load weather CSV
16 weather_df = spark.read.csv(weather_path, header=True, inferSchema=True)
```

The code is annotated with line numbers and includes comments explaining the data sources and target path.

Run the job

The screenshot shows the AWS Glue Studio monitoring page for the job 'Job Run - jr_ffdcf0095b9a0f673923154675776a3fb7edb269502fdb23c7ff5de89f774f16'. The left sidebar is identical to the previous screenshot. The main area shows the 'Run details' section with the following information:

Run details		Info	
jr_ffdcf0095b9a0f673923154675776a3fb7edb269502fdb23c7ff5de89f774f16			
Job name	Id	Run status	Glue version
taxi-weather-join-job	jr_ffdcf0095b9a0f673923154675776a3fb7edb269502fdb23c7ff5de89f774f16	Succeeded	4.0
Retry attempt number	Start time (Local)	End time (Local)	Start time (UTC)
Initial run	08/25/2025 16:44:46	08/25/2025 16:46:38	2025/08/25 20:44:46
End time (UTC)	Start-up time	Execution time	Last modified on (Local)
2025/08/25 20:46:38	19 seconds	1 minute 33 seconds	08/25/2025 16:46:38
Last modified on (UTC)	Trigger name	Security configuration	Timeout
2025/08/25 20:46:38	-	-	2880 minutes
Max capacity	Number of workers	Worker type	Execution class
10 DPU	10	G.1X	Standard
Log group name	Cloudwatch logs	Performance and debugging recommendations	Usage profile
/aws-glue/jobs	<ul style="list-style-type: none"> Output logs Error logs 	<ul style="list-style-type: none"> View in CloudWatch 	-

File created successfully in s3 after job run

The screenshot shows the AWS S3 console interface. The left sidebar lists 'General purpose buckets' such as Directory buckets, Table buckets, Vector buckets, Access Grants, Access Points (General Purpose Buckets, Fsx file systems), Access Points (Directory Buckets), Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. The main area is titled 'joined' and shows 'Objects (1)'. A single object is listed:

Name	Type	Last modified	Size	Storage class
part-00000-d1467ead-e630-4ff1-81c9-d61a24fe504-c000.snappy.parquet	parquet	August 25, 2025, 16:46:28 (UTC-04:00)	23.7 KB	Standard

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
```

```
D:\public-transport-demand>aws glue create-crawler --name taxi-hourly-crawler --role AWSGlueServiceRole-Transport --database-name transport_analytics --targ  
ets "{\"S3Targets\":[{\\"Path\\\":\"s3://transportation-project-processed/taxi/hourly/\\"}]}" --region us-east-1 && aws glue start-crawler --name taxi-hourly-cr  
awler --region us-east-1  
D:\public-transport-demand>
```

AWS Glue > Tables > hourly

Last updated: August 25, 2025 at 21:06:11

Advanced properties

Schema (5)

#	Column name	Data type	Partition key	Comment
1	date	date	-	-
2	hour	int	-	-
3	dayofweek	int	-	-
4	avg_fare	double	-	-
5	trip_count	bigint	-	-
6	avg_distance	double	-	-

- 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
```

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

AWS Glue > Tables > joined

Schema (10)

#	Column name	Data type	Partition key	Comment
1	date	date	-	-
2	hour	int	-	-
3	dayofweek	int	-	-
4	trip_count	bigint	-	-
5	avg_fare	double	-	-
6	avg_distance	double	-	-
7	temperature	double	-	-
8	humidity	double	-	-
9	pressure	double	-	-
10	wind_speed	double	-	-

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

Athena Queries

Key Queries:

1. Validate joined dataset:

```
SELECT COUNT(*) FROM transport_analytics.joined;
```

The screenshot shows two views of the Amazon Athena Query Editor interface. The top view displays the 'Query editor tabs' section with a single query named 'Query 1'. The query itself is:

```
1 SELECT COUNT(*)
2 FROM transport_analytics.joined;
```

The bottom view shows the results of the query execution. The sidebar on the left lists tables and views under the 'transport_analytics' catalog. The main area shows the query results:

#	_col0
1	744

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 :

```
1  SELECT
2    CASE
3      WHEN humidity < 40 THEN 'Low'
4      WHEN humidity BETWEEN 40 AND 70 THEN 'Medium'
5      ELSE 'High'
6    END AS humidity_band,
7    AVG(trip_count) AS avg_trips
8  FROM transport_analytics.joined
9  GROUP BY
10   CASE
11     WHEN humidity < 40 THEN 'Low'
12     WHEN humidity BETWEEN 40 AND 70 THEN 'Medium'
13     ELSE 'High'
14   END
15   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

#	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;
```

The screenshot shows the AWS Lambda function configuration page. At the top, there are tabs for 'Basic' (selected), 'Code', 'Environment', 'Logs', and 'Metrics'. Below the tabs, there are sections for 'Lambda function' (Name: 'lambda-1', Region: 'us-east-1'), 'Runtime' (Node.js 14.x), and 'Handler' (index.handler). Under the 'Configuration' section, there is a 'Timeout' dropdown set to '300 seconds'. The 'Code' tab displays the Lambda function code in JSON format:

```
{ "version": "2018-06-01", "functionName": "lambda-1", "runtime": "nodejs14.x", "handler": "index.handler", "timeout": 300 }
```

The 'Logs' tab shows log events with timestamps from 2023-09-11T10:45:00Z to 2023-09-11T10:45:00Z. The 'Metrics' tab shows CloudWatch Metrics for the function.

4. Day-of-Week Demand Patterns

The screenshot shows two consecutive views of the Amazon Athena Query Editor interface.

Query Editor View:

- Data Source:** AwsDataCatalog
- Catalogue:** None
- Database:** transport_analytics
- Tables and views:** Joined
- Query 1:**

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

Query Results View:

- Views (0):** Joined
- Query results:** Completed
- Time in queue:** 112 ms **Run time:** 466 ms **Data scanned:** 3.23 KB
- Results (7):**

#	dayofweek	avg_trips
1	1	16490.604166666668
2	2	13884.072916666666
3	3	14311.6875
4	4	17541.53125
5	5	18068.441666666666
6	6	18559.891666666666
7	7	19830.9

5. Demand vs Pressure

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Data' panel is open, displaying configuration for a data source (AwsDataCatalog), catalogue (None), and database (transport_analytics). Below this, the 'Tables and views' section lists two tables: 'hourly' and 'joined'. On the right, the 'Query 1' editor contains the following SQL query:

```

1 SELECT
2     ROUND(pressure/5)*5 AS pressure_band,
3     AVG(trip_count) AS avg_trips
4 FROM transport_analytics.joined
5 GROUP BY ROUND(pressure/5)*5
6 ORDER BY pressure_band;
7
8

```

Below the query editor, the status bar indicates 'SQL Ln 8, Col 1'. At the bottom of the editor are buttons for 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'.

The screenshot shows the Amazon Athena Query Editor interface with the 'Query results' tab selected. The status bar at the top indicates the query was completed. The results table displays 12 rows of data:

#	pressure_band	avg_trips
1	995.0	19839.714285714286
2	1000.0	12059.21052631579
3	1005.0	14407.441176470587
4	1010.0	16626.981308411214
5	1015.0	13989.309734513274
6	1020.0	17768.72067039106
7	1025.0	19828.97391304348
8	1030.0	21520.457142857143
9	1035.0	15208.019607843138

At the bottom of the results table are buttons for 'Copy' and 'Download results CSV'.

6. Combined Effect: Hour + Temperature Band

The screenshot shows two screenshots of the Amazon Athena Query Editor interface.

Query Editor (Left Screenshot):

- Data Source:** AwsDataCatalog
- Catalogue:** None
- Database:** transport_analytics
- Tables and views:** transport_analytics.joined

Query 1:

```
1 SELECT
2     hour,
3     CASE
4         WHEN (temperature - 273.15) < 0 THEN 'Freezing'
5         WHEN (temperature - 273.15) BETWEEN 0 AND 10 THEN 'Cold'
6         WHEN (temperature - 273.15) BETWEEN 10 AND 20 THEN 'Warm'
7     END AS temp_band,
8     AVG(trip_count) AS avg_trips
9     FROM transport_analytics.joined
10    GROUP BY hour,
11    CASE
12        WHEN (temperature - 273.15) < 0 THEN 'Freezing'
13        WHEN (temperature - 273.15) BETWEEN 0 AND 10 THEN 'Cold'
14        WHEN (temperature - 273.15) BETWEEN 10 AND 20 THEN 'Warm'
15    END
```

Query Results (Right Screenshot):

Completed (Time in queue: 107 ms, Run time: 450 ms, Data scanned: 7.17 KB)

#	hour	temp_band	avg_trips
1	0	Cold	15712.0
2	0	Freezing	15054.26923076923
3	1	Cold	12097.0
4	1	Freezing	11333.076923076924
5	2	Cold	5952.6
6	2	Freezing	9168.076923076924
7	3	Cold	4780.0
8	3	Freezing	6820.5
9	3	Warm	2071.0

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

	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	bea27d66-98a6-47b9-9425-6149f474ee34.csv	csv	August 25, 2025, 20:23:51 (UTC-04:00)	22.0 B	Standard
<input type="checkbox"/>	bea27d66-98a6-47b9-9425-6149f474ee34.csv.metadata	metadata	August 25, 2025, 20:23:52 (UTC-04:00)	83.0 B	Standard
<input type="checkbox"/>	c2237499-2458-41a4-8902-66095816b715.csv	csv	August 25, 2025, 19:51:42 (UTC-04:00)	22.0 B	Standard
<input type="checkbox"/>	c2237499-2458-41a4-8902-66095816b715.csv.metadata	metadata	August 25, 2025, 19:51:42 (UTC-04:00)	83.0 B	Standard
<input type="checkbox"/>	Unsaved/	Folder	-	-	-

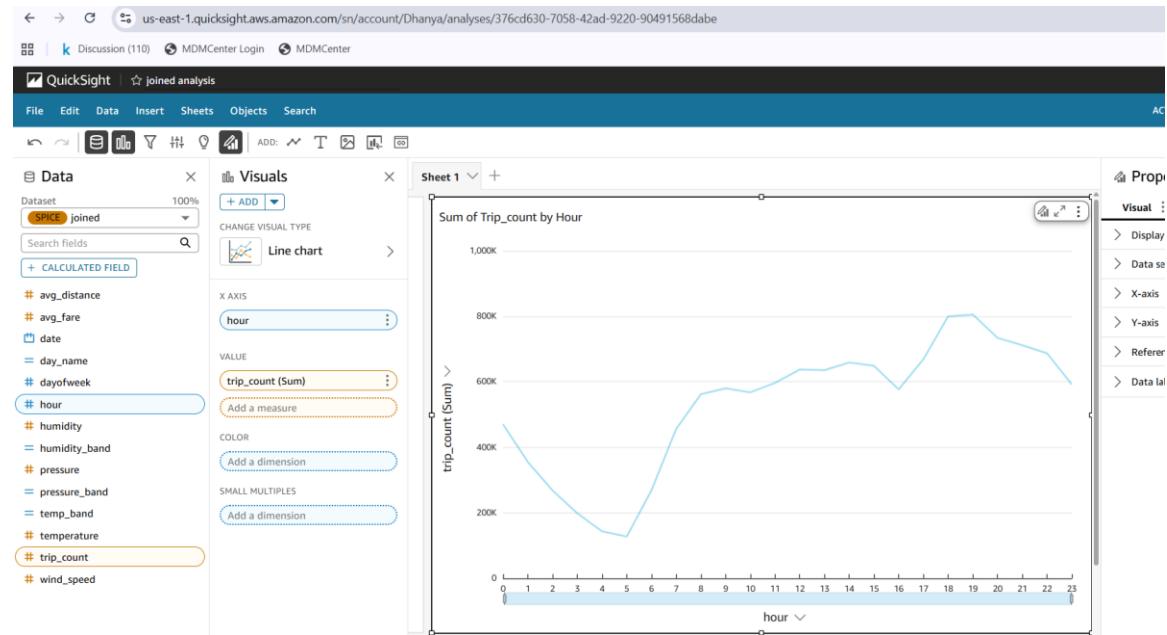
QuickSight Dashboards

Steps:

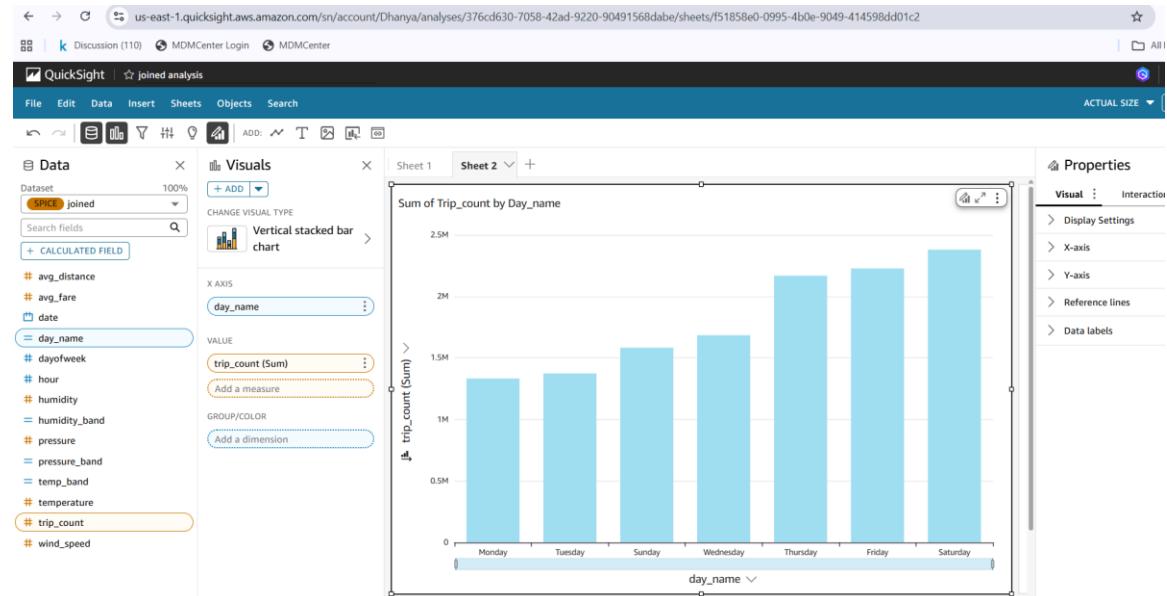
1. Enabled QuickSight with Athena integration.
2. Granted QuickSight access to S3 + Glue Data Catalog.
3. Created dataset from Athena table taxi_weather_joined.
4. Imported into SPICE for performance.

Dashboards Created:

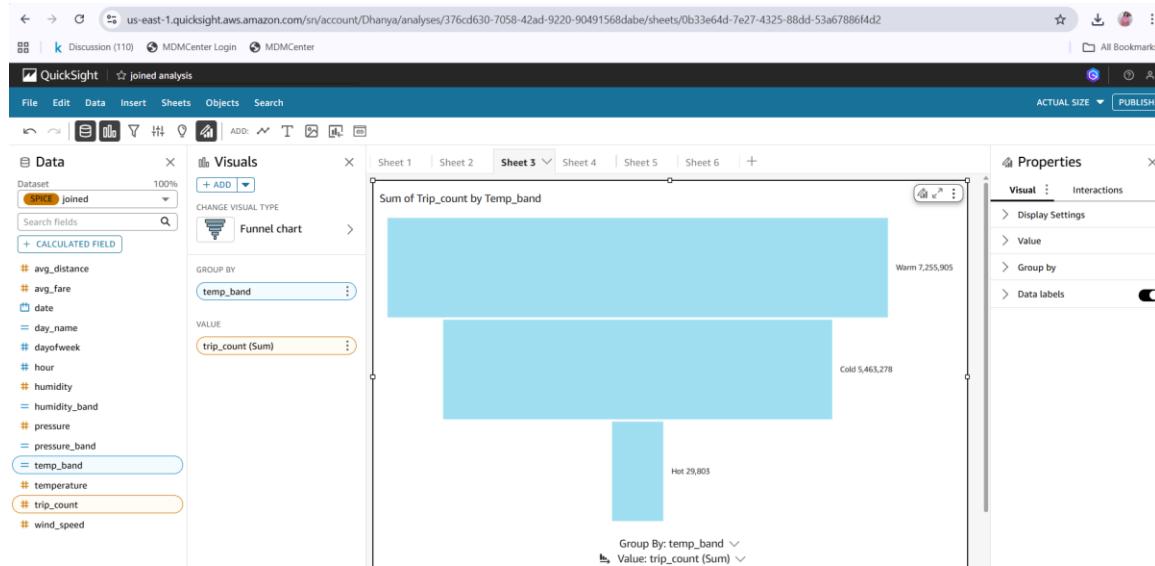
1. Trips by Hour (Line Chart).



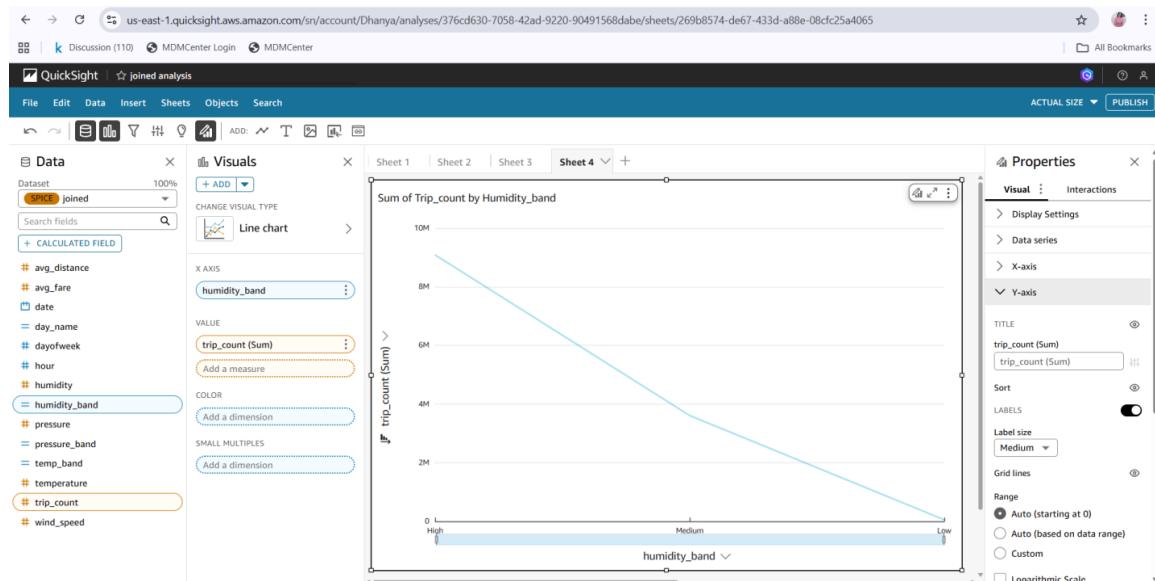
2. Trips by Day of Week (Bar).



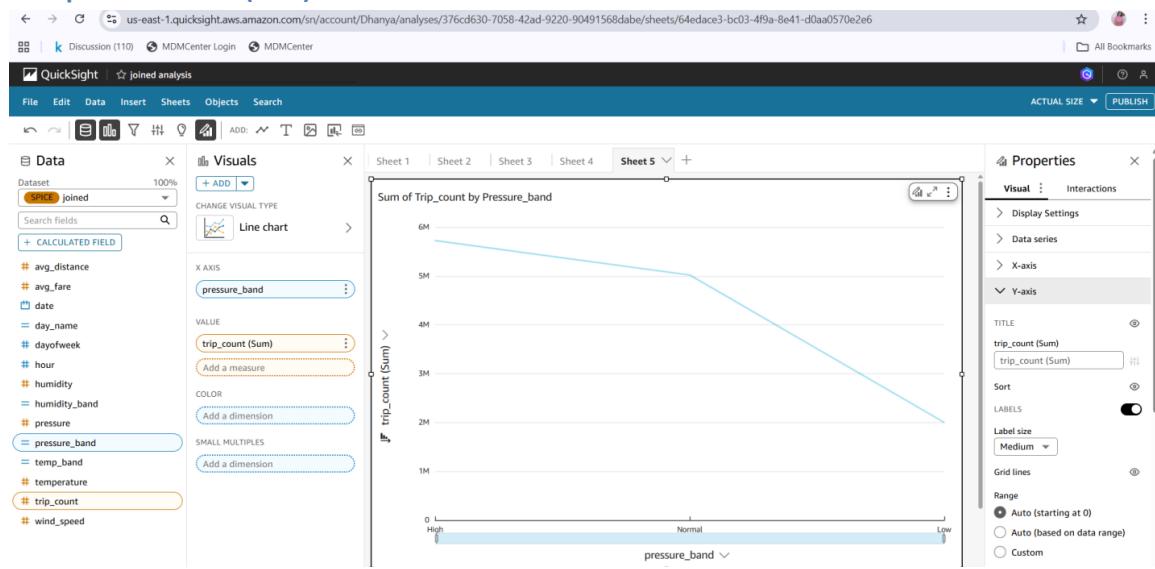
3. Trips vs Temperature Band (Funnel/Bar).



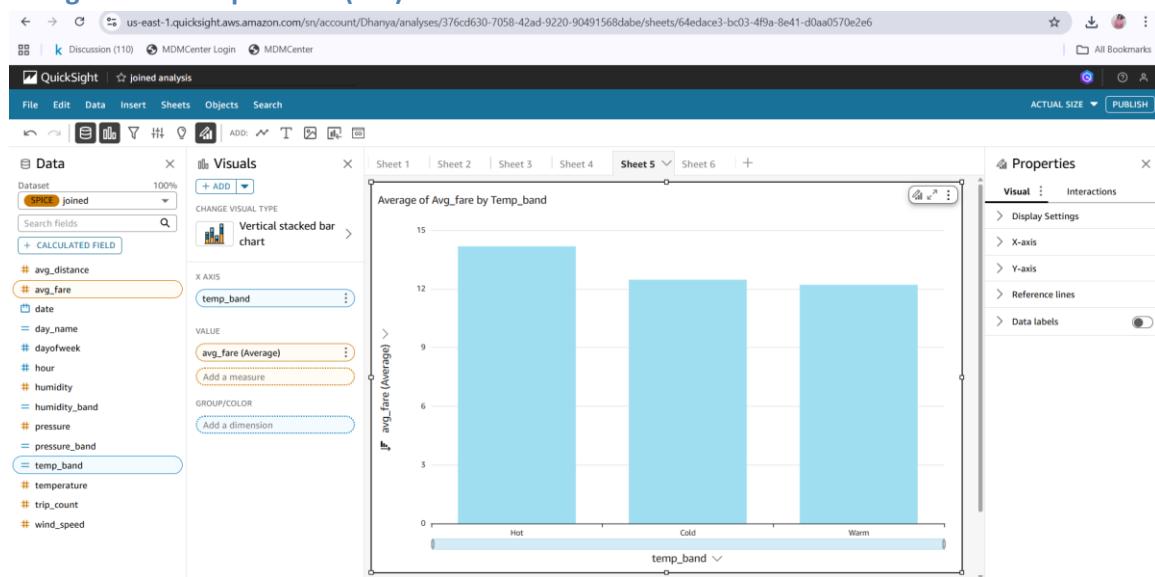
4. Trips vs Humidity (Line).



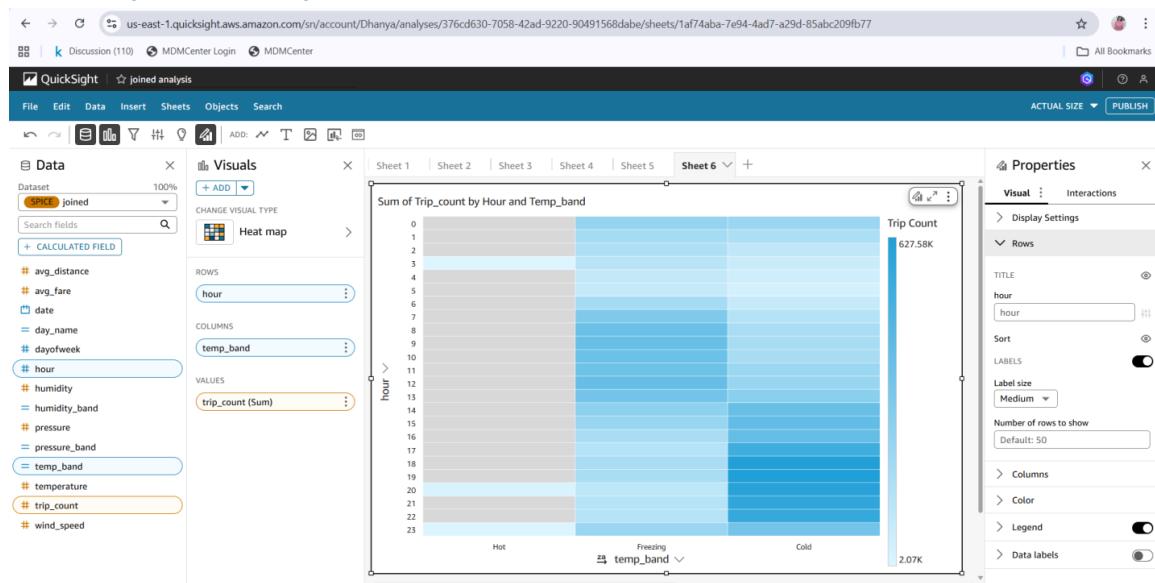
5. Trips vs Pressure (Line).



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.

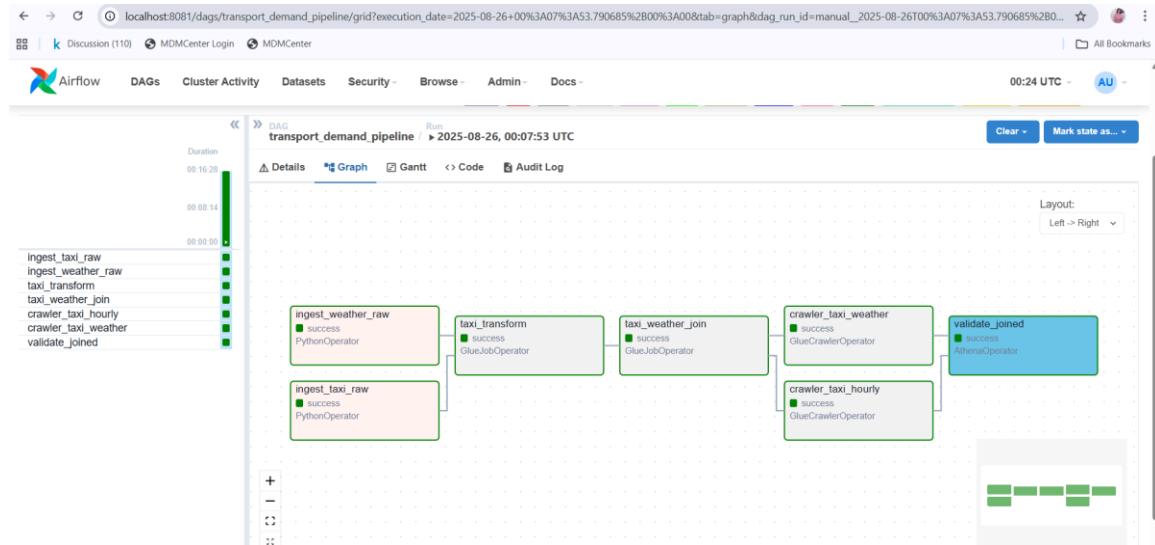
Airflow Orchestration

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.

The image shows two screenshots of the Airflow web interface. The top screenshot is a 'Sign In' page with fields for 'Username' (admin) and 'Password' (left blank). The bottom screenshot is the 'DAGs' list page, showing one DAG named 'transport_demand_pipeline'. This DAG has three tags: 'athena', 'glue', and 'transport'. It was last run on 2025-08-24 at 00:00:00 and has 12 tasks listed under 'Recent Tasks'.



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>