🎓 📓 Final Project Documentation – Real-Time Traffic Analytics & Smart Lighting System

Project Title: Smart Traffic Management System + Solar Powered Street Lighting
Team Name: *Smart City Innovators*
Course: DEPI – Data Engineering Program
Date: 2025
Team Members:

- Hassan Gamal Ghanem
- Maya Yaser Amin
- Mohammed Mohammed Sobhy
- Amr Mohammed Youssef
- Radwa Hany Sobhy
- Habiba Ashraf Elboghdady

1. Executive Summary

This project delivers a full **end-to-end real-time analytics platform** that processes live traffic sensor data using **IoT + MQTT + Kafka + Spark Streaming + SQL Server + SSIS + Parquet Data Lake**, combined with a smart lighting actuator and alert notification system.

The solution enhances **traffic monitoring**, **energy optimization**, and **situational awareness** in smart cities.

2. Problem Statement

Cities face:

- Traffic congestion

- Energy waste in street lighting
- Lack of real-time situational awareness
- Slow decision-making due to batch-based systems

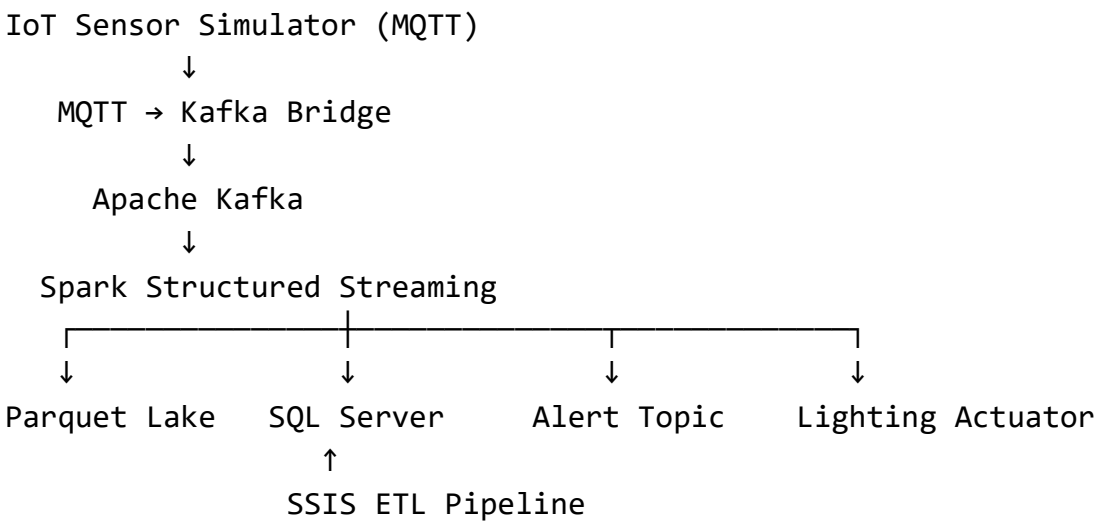
Our system solves this using **real-time streaming analytics**, **automated alerts**, and **intelligent lighting control**.

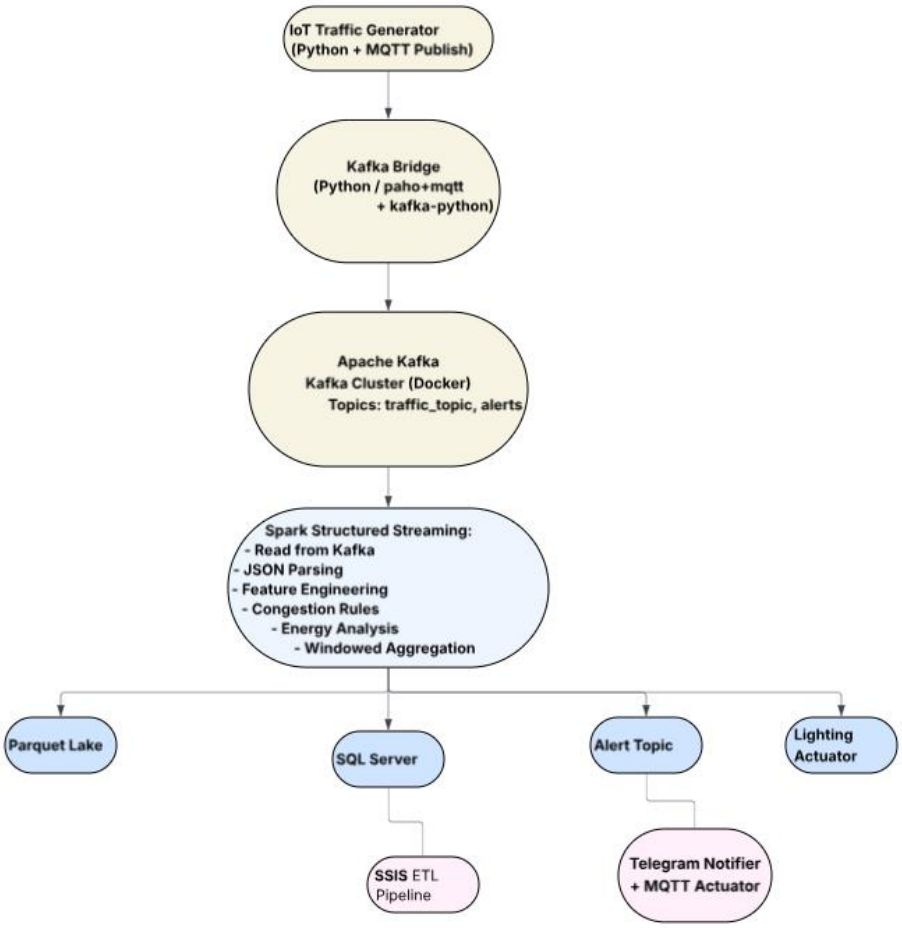
3. System Goals

- Receive IoT sensor traffic data in real-time.
- Stream, transform, and aggregate data continuously.
- Detect congestion & energy overload situations.
- Store processed data for BI & analytics.
- Control smart lighting automatically.
- Provide dashboards & notifications.

4. High-Level Architecture

Diagram (describe or I generate PNG later):





Main Components:

- MQTT Sensor Publisher
- MQTT–Kafka Bridge
- Dockerized Kafka + Zookeeper
- Spark Streaming Engine (aggregation + alerts)
- SQL Server (3-table normalized schema)
- Parquet Data Lake
- SSIS automated ETL
- Lighting Actuator (MQTT)
- Telegram Alerts Bot

5. Detailed System Design

5.1 MQTT Sensor Data Generator

Produces fields:

- timestamp
- street_name
- vehicle_speed
- vehicle_count
- weather
- traffic_light
- solar_energy
- lighting_demand

Configured via YAML & runs continuously.

5.2 MQTT → Kafka Bridge

Purpose: unify IoT → Kafka ingestion channel

Key snippet:

```
def on_message(client, userdata, msg):
    data = json.loads(msg.payload.decode())
    producer.send("traffic_topic", value=data)
```

5.3 Kafka Cluster (Docker-Compose)

Important services:

- Zookeeper
- Kafka broker (advertised listeners)
- Jupyter Spark Notebook

5.4 Spark Structured Streaming Pipeline

Input → Transform → Output

Parsing Kafka Stream:

```
df = spark.readStream.format("kafka")\
  .option("subscribe", "traffic_topic").load()
```

Processing:

- time parsing
- peak hour flag
- congestion classification
- energy consumption model
- watermarks & window aggregation

Alerts:

```
alerts_for_kafka = agg2.filter(col("is_alert")==True)
```

5.5 Storage Layer

A) Parquet Data Lake

Optimized for:

- ML Training
- Time-series analytics
- Columnar compression

B) SQL Server Split Tables

Spark writes each micro-batch into three tables:

1. traffic_sensors_data
2. traffic_weather_conditions
3. traffic_energy_analysis

5.6 SSIS ETL Package

Responsibilities:

- Restore `.bak` database
- Create missing tables
- Load CSVs from Spark
- Bulk Insert into SQL Server
- Archive loaded files

Includes:

- Execute SQL Task (RESTORE)
- Data Flow (CSV → OLE DB)
- Script Task (bulk insert)
- File Processing automation
- Failure logging

6. Data Model & Database Design

6.1 📌 ER TEXT (Entity–Relationship Textual Description)

This document describes all entities and the relationships between them in a fully textual ERD format.

◇ Entity 1: traffic_sensors_data

Attributes:

- timestamp (PK)
- street_name (PK)
- vehicle_count
- vehicle_speed
- is_congested

- congestion_level
- is_peak_hour
- day_of_week

◇ Entity 2:traffic_weather_conditions

Attributes:

- timestamp (PK)
- street_name (PK)
- weather
- light_level
- traffic_light

◇ Entity 3: traffic_energy_analysis

Attributes:

- timestamp (PK)
- street_name (PK)
- solar_energy_level
- lighting_demand
- lighting_consumption_kWh
- energy_alert_flag

◇ Entity 4: traffic_sensors_dacr

Attributes:

- timestamp (PK)
- street_name (PK)
- vehicle_count
- avg_speed
- hour_of_day

- day_of_week
- is_weekend
- is_peak
- congestion_flag

◇ Entity 5: lighting_controller

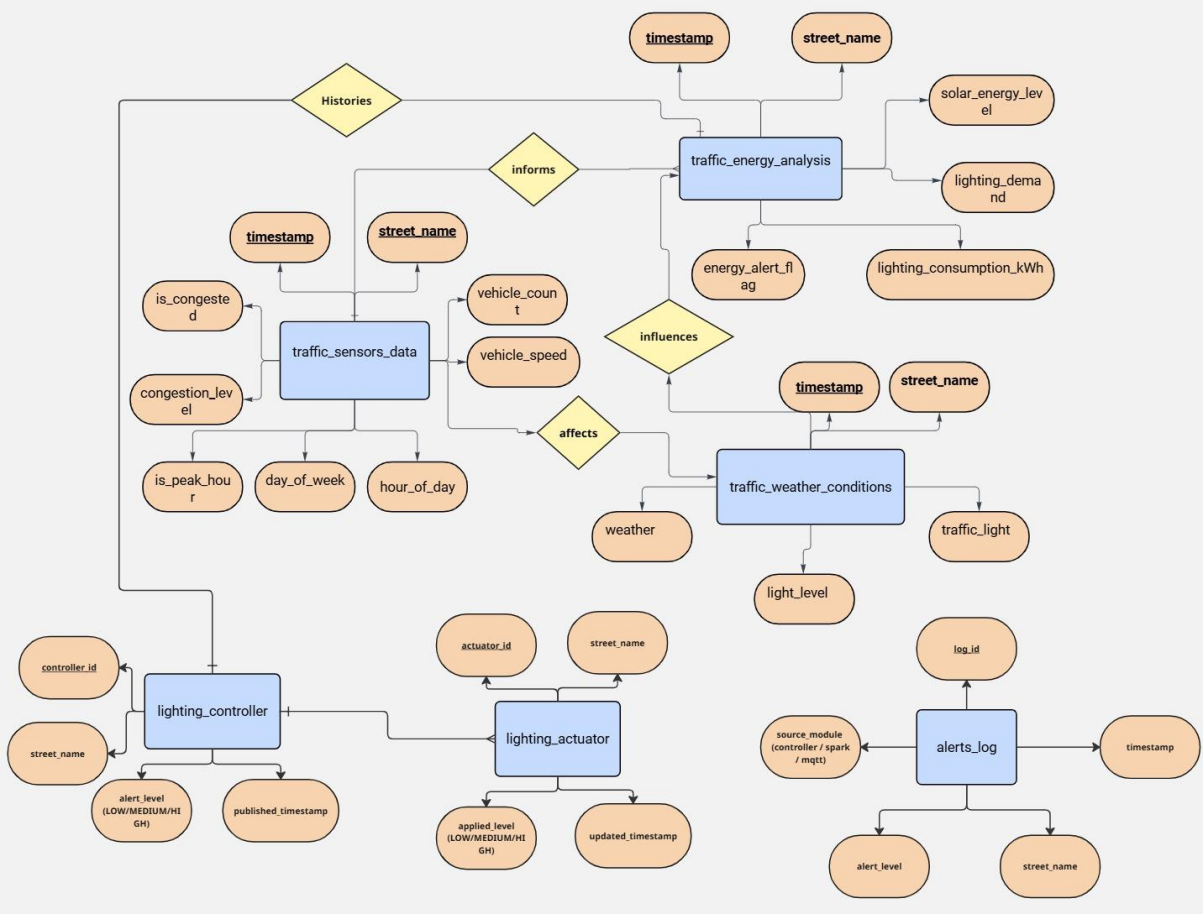
Attributes:

- control_id (PK)
- street_name
- lighting_level
- decision_reason
- timestamp

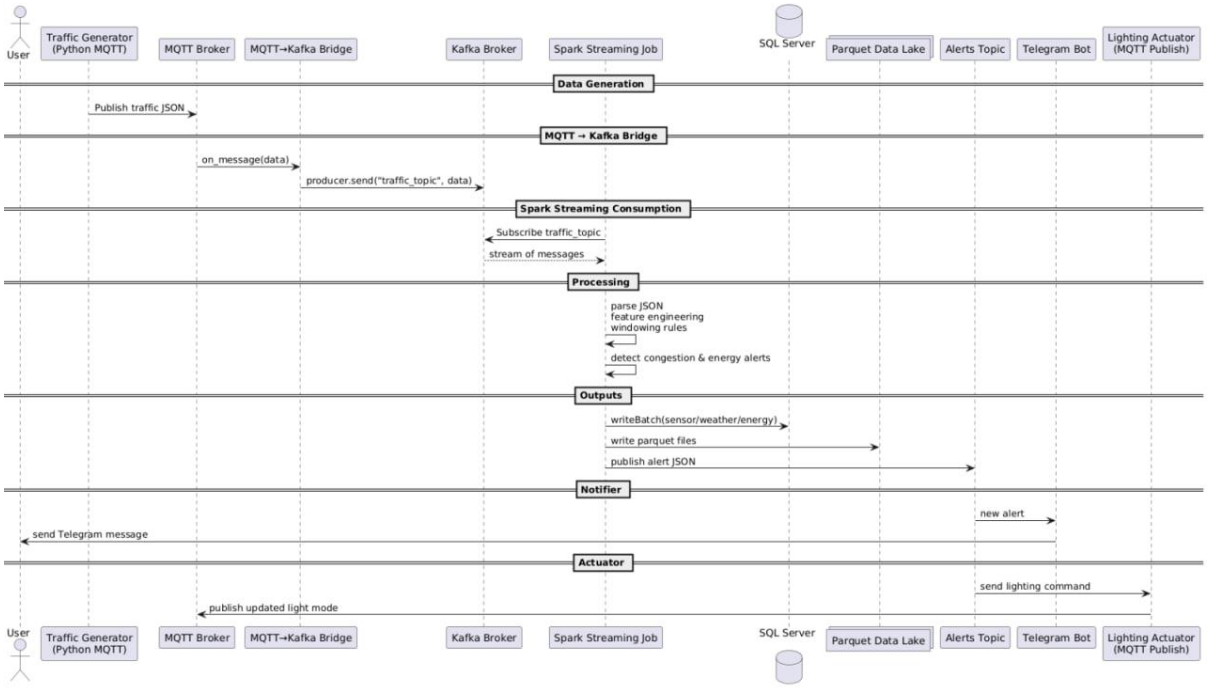
◇ Entity 6: lighting_actuator

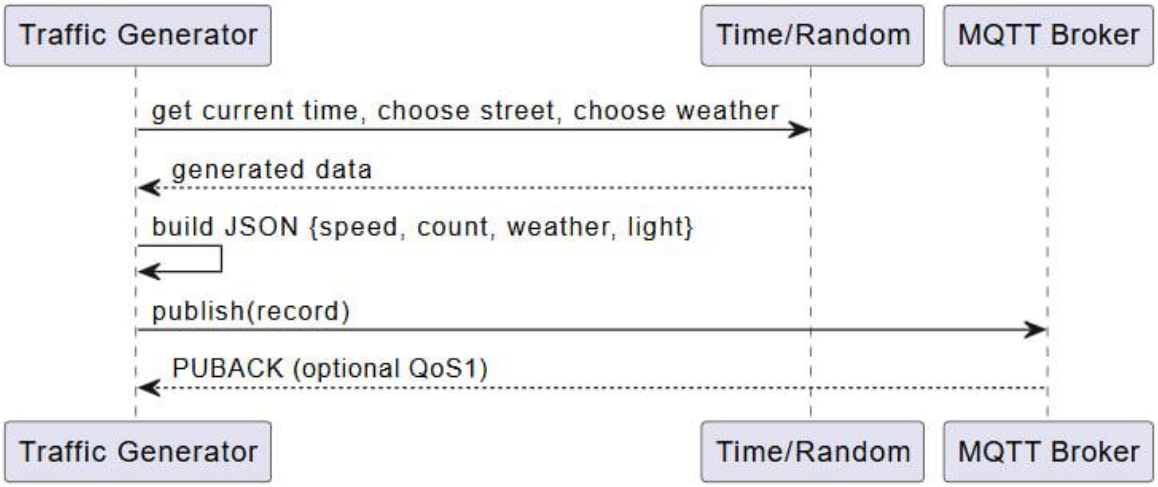
Attributes:

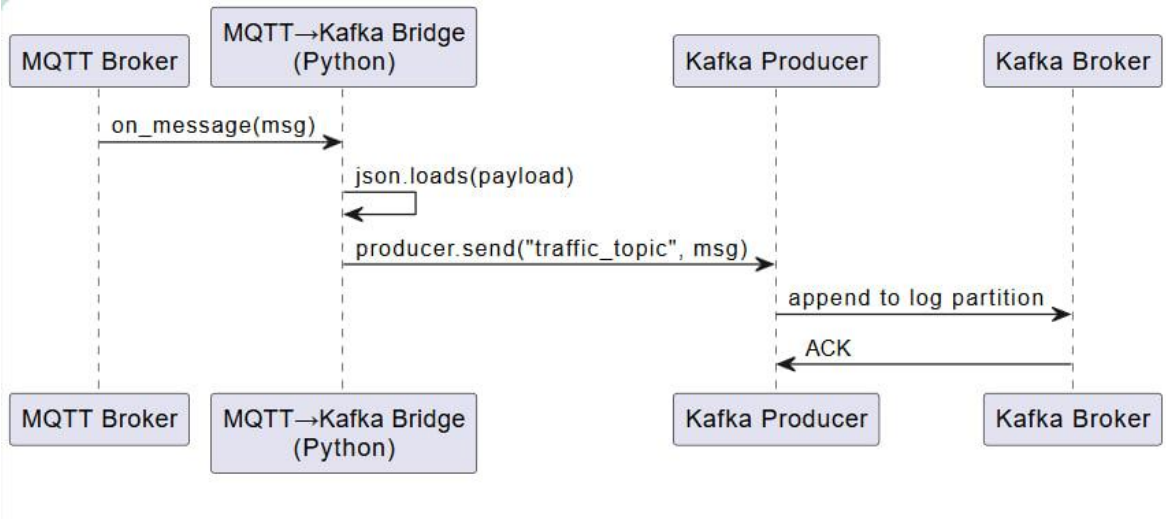
- action_id (PK)
- street_name
- applied_level
- controller_timestamp

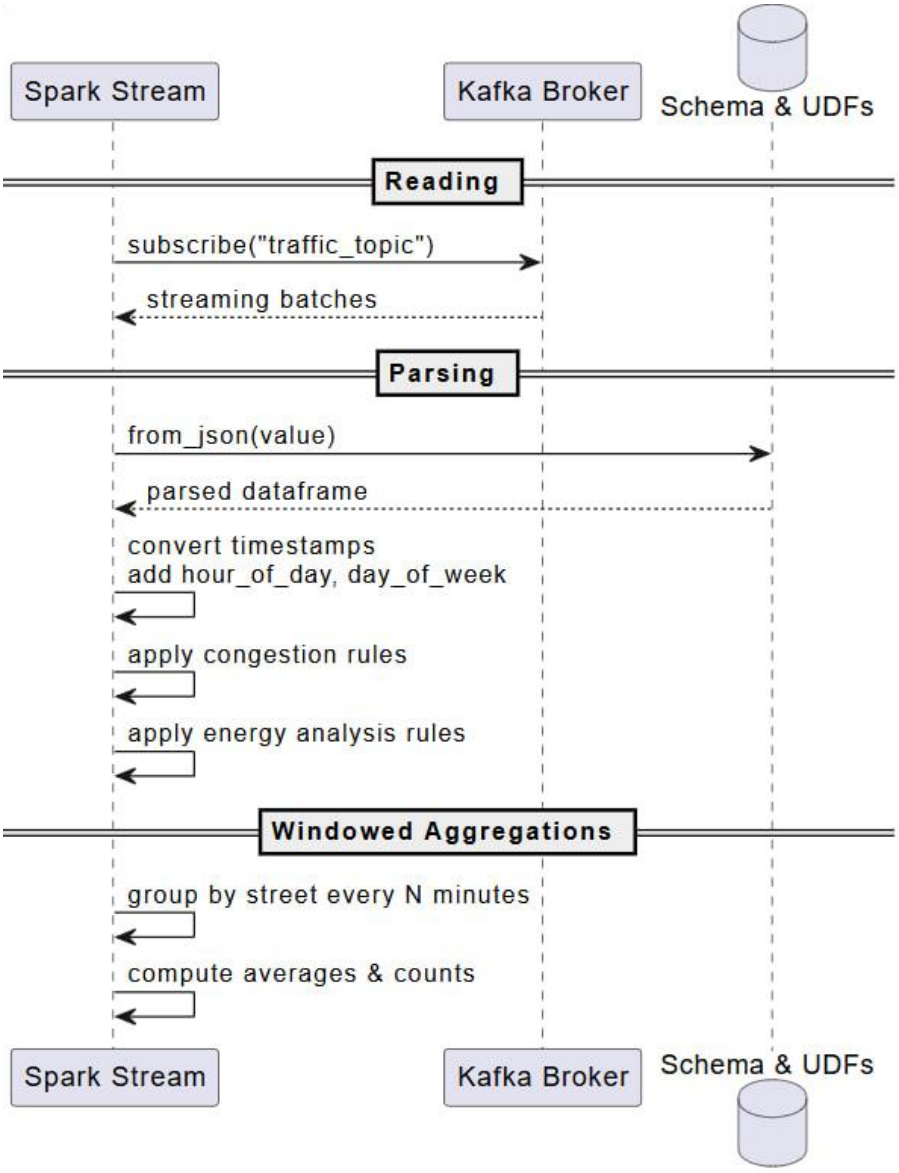


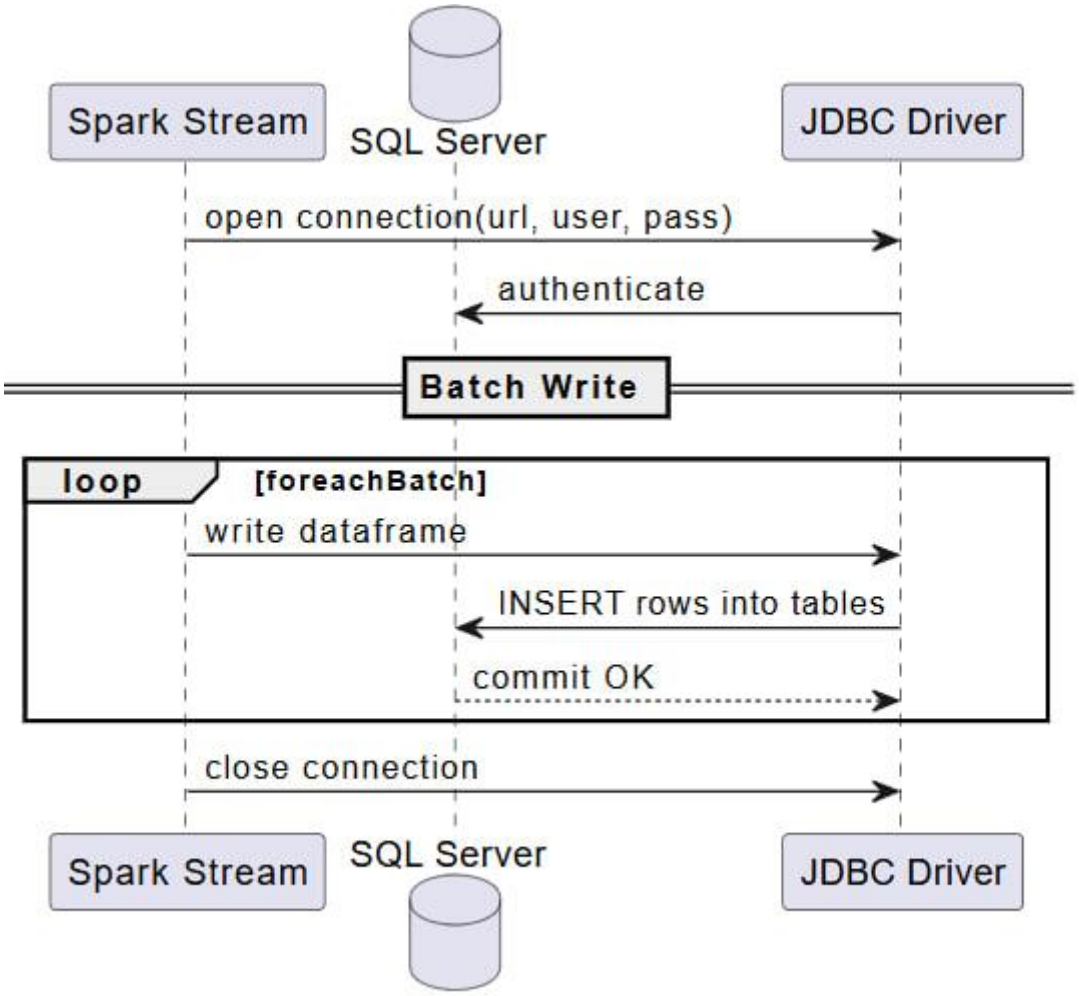
6.2 sequence diagrams:

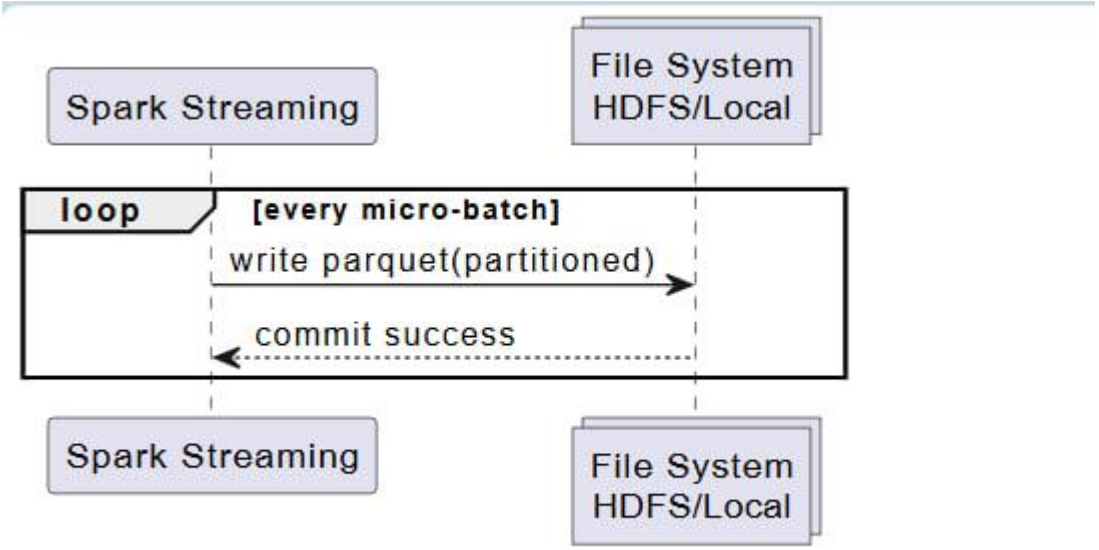


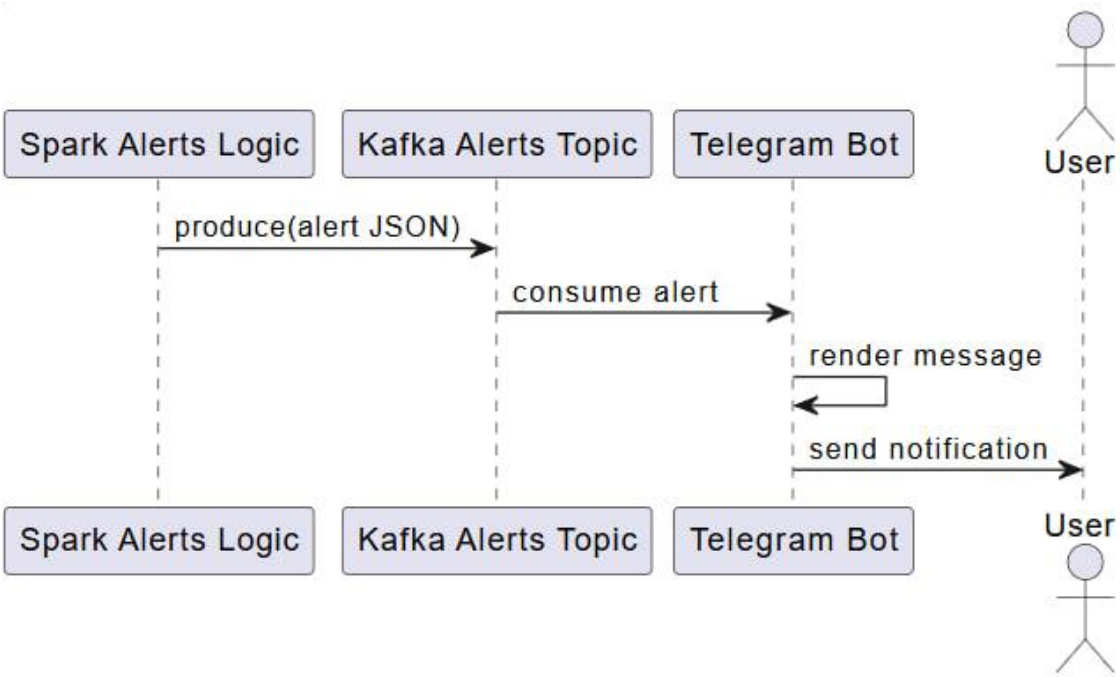

Traffic Generator
Time/Random
MQTT Broker
get current time, choose street, choose weather
generated data
build JSON {speed, count, weather, light}
publish(record)
PUBACK (optional QoS1)
Traffic Generator
Time/Random
MQTT Broker

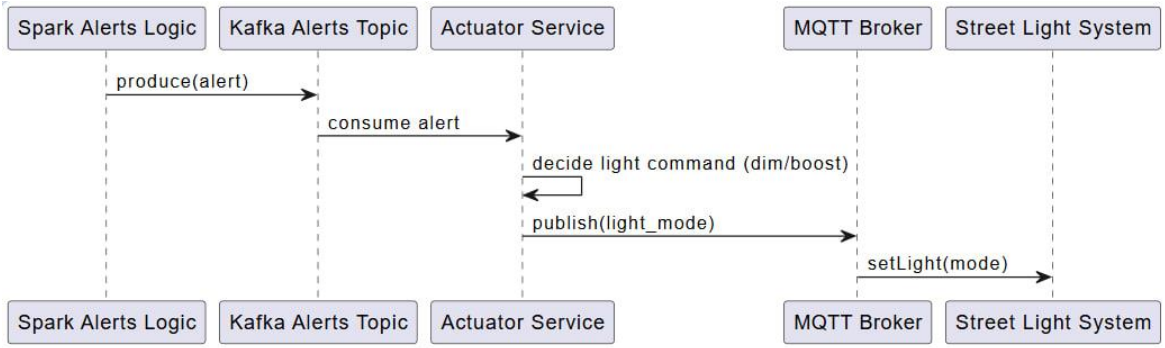

MQTT Broker
MQTT→Kafka Bridge (Python)
Kafka Producer
Kafka Broker
on_message(msg)
json.loads(payload)
producer.send("traffic_topic", msg)
append to log partition
ACK
MQTT Broker
MQTT→Kafka Bridge (Python)
Kafka Producer
Kafka Broker


Spark Stream
Kafka Broker
Schema & UDFs
Reading
subscribe("traffic_topic")
streaming batches
Parsing
from_json(value)
parsed dataframe
convert timestamps
add hour_of_day, day_of_week
apply congestion rules
apply energy analysis rules
Windowed Aggregations
group by street every N minutes
compute averages & counts
Spark Stream
Kafka Broker
Schema & UDFs

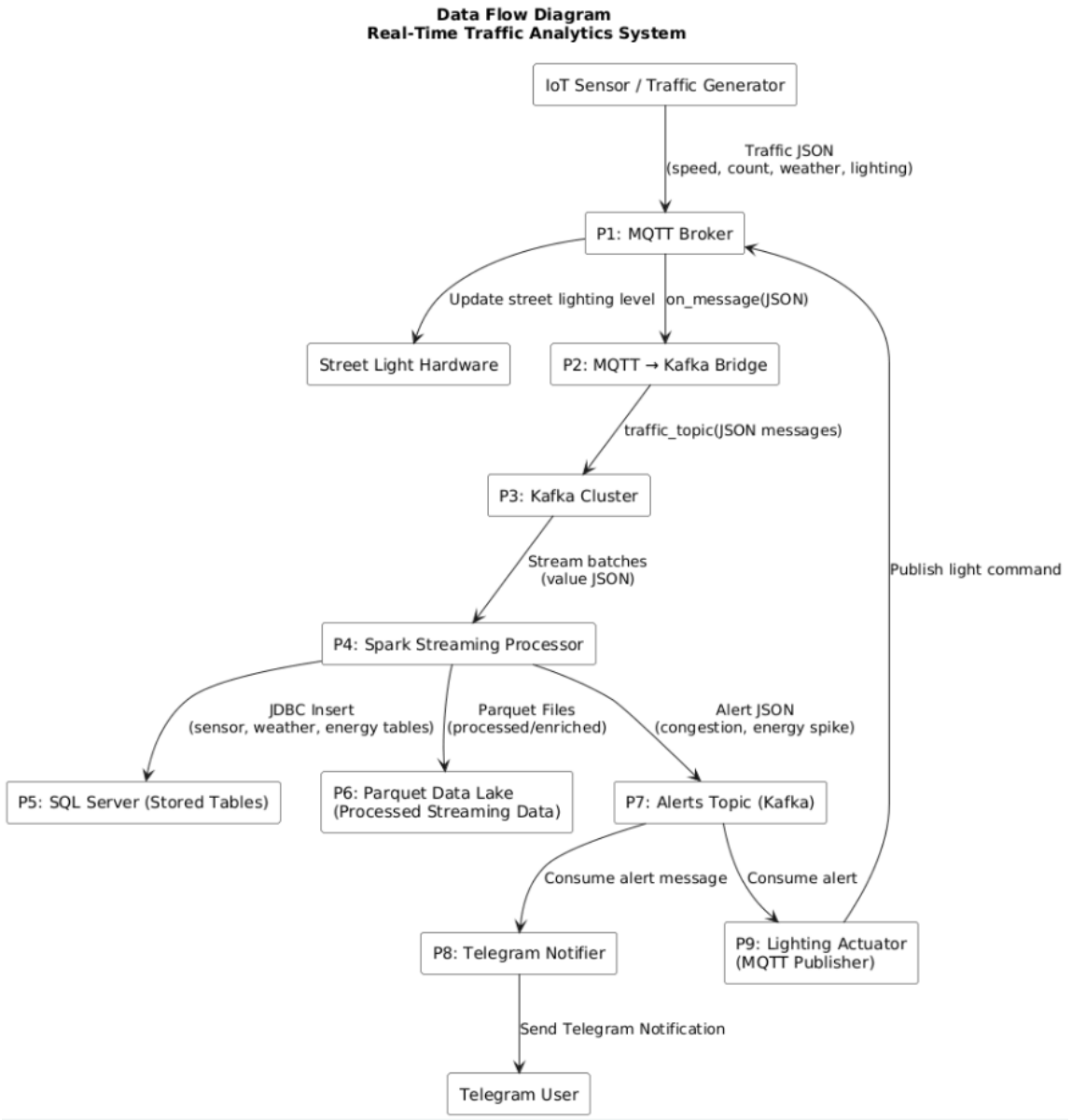

Spark Stream
SQL Server
JDBC Driver
open connection(url, user, pass)
authenticate
Batch Write
loop
[foreachBatch]
write dataframe
INSERT rows into tables
commit OK
close connection
Spark Stream
SQL Server
JDBC Driver


Spark Streaming
File System
HDFS/Local
loop
[every micro-batch]
write parquet(partitioned)
commit success
Spark Streaming
File System
HDFS/Local





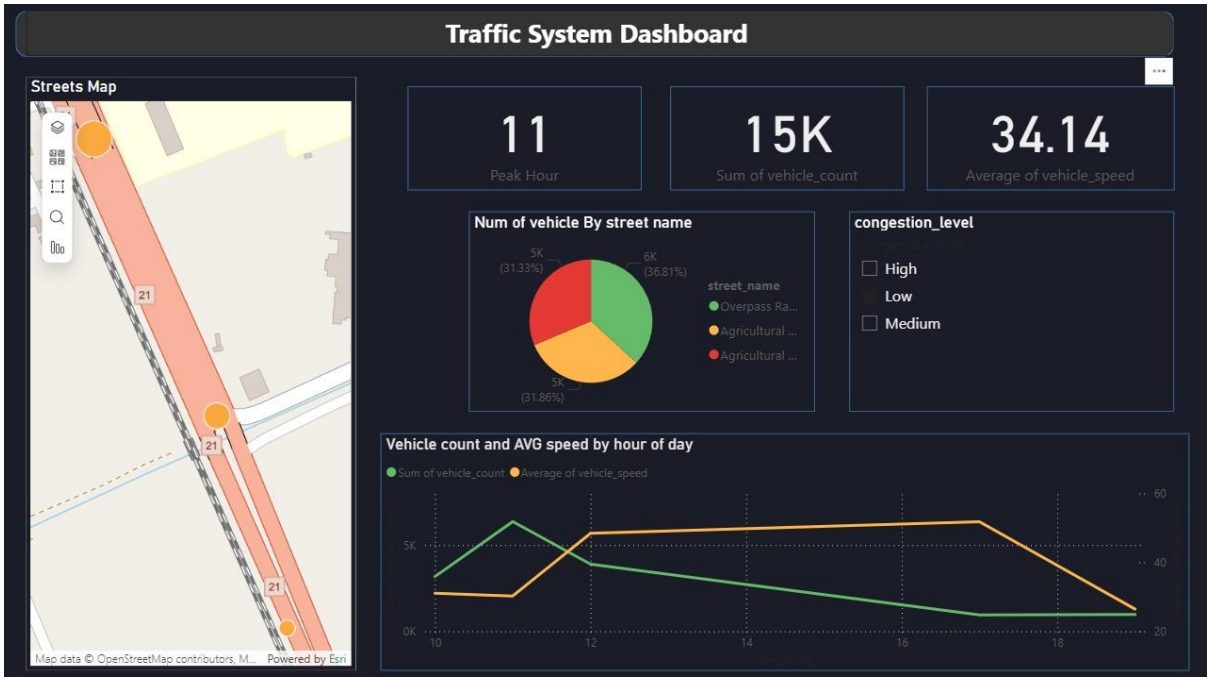
6.3 Data Flow Digram:



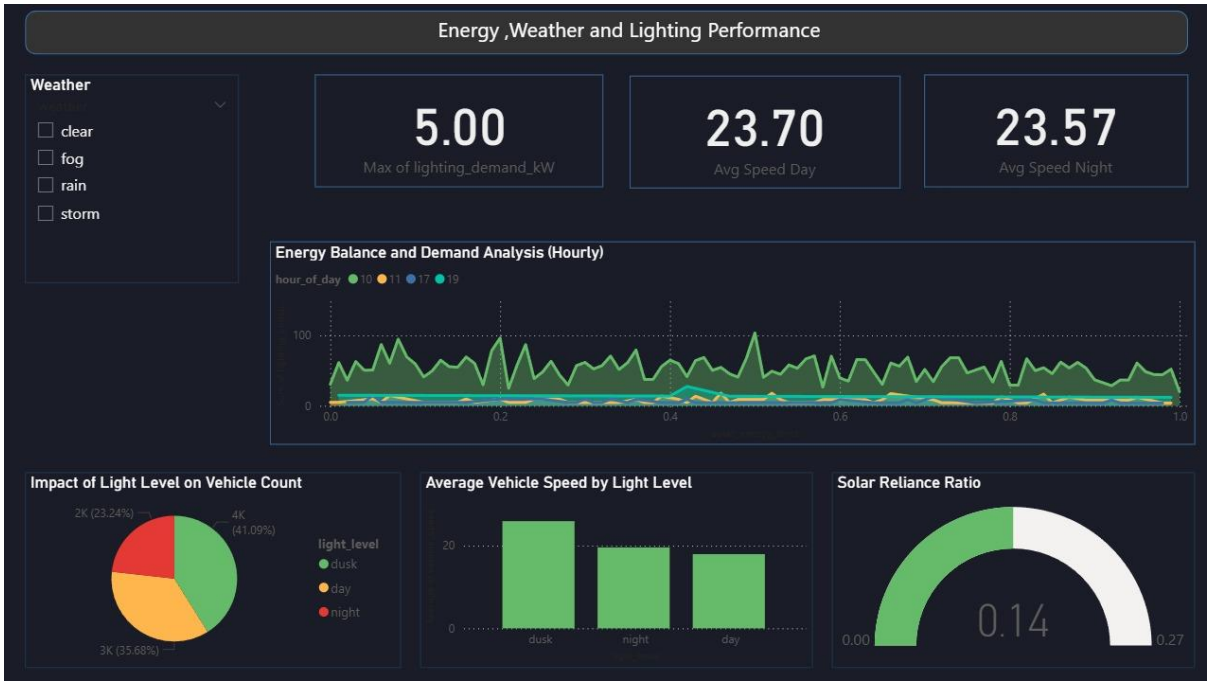
7. Wireframes (UI / Dashboard Mockups)

Dashboard Sections:

- Real-time Traffic Status
- Congestion Heatmap
- Energy Usage Chart
- Alerts Timeline
- Per-Street Metrics Viewer
- Historical Parquet Explorer

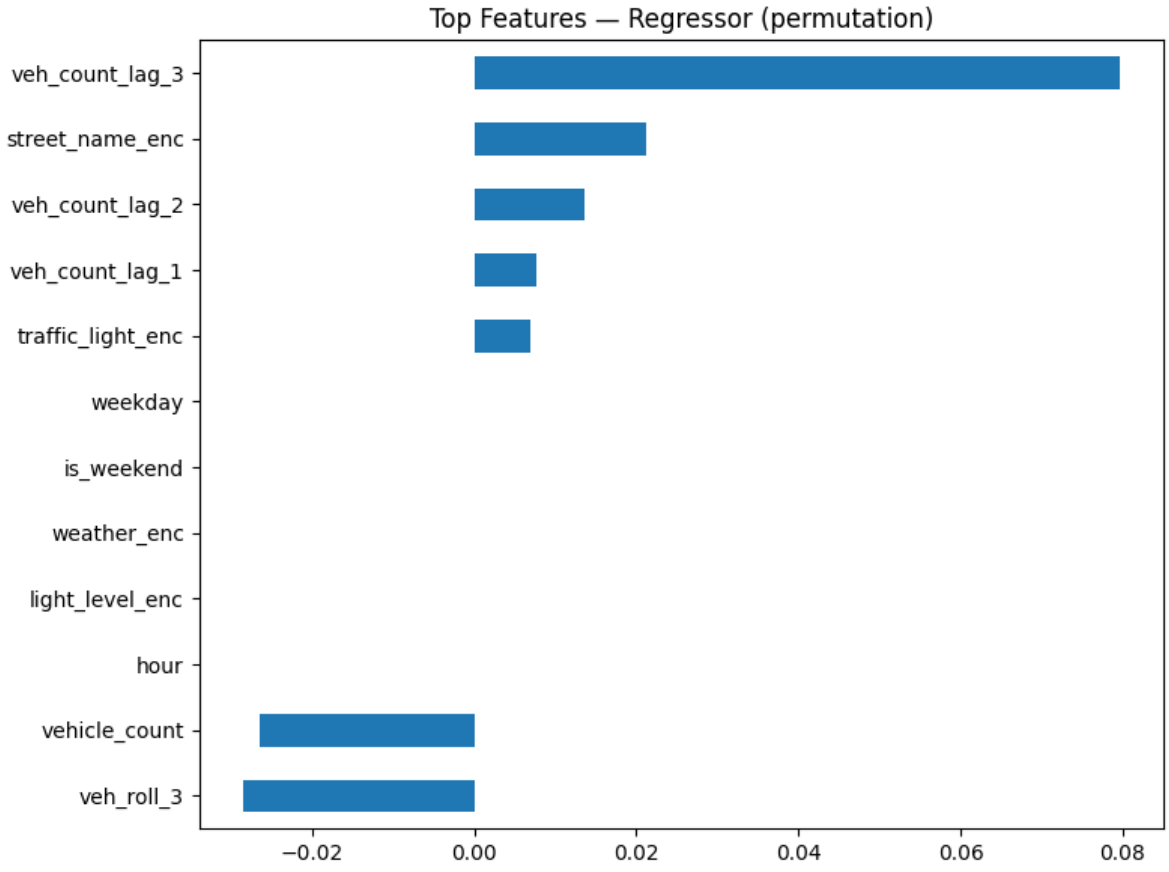

Traffic System Dashboard
Streets Map
11
Peak Hour
15K
Sum of vehicle_count
34.14
Average of vehicle_speed
Num of vehicle By street name
5K (31.33%)
6K (36.81%)
5K (31.86%)
street_name
Overpass Ra...
Agricultural ...
Agricultural ...
congestion_level
High
Low
Medium
Vehicle count and AVG speed by hour of day
Sum of vehicle_count
Average of vehicle_speed
Map data © OpenStreetMap contributors, M...
Powered by Esri

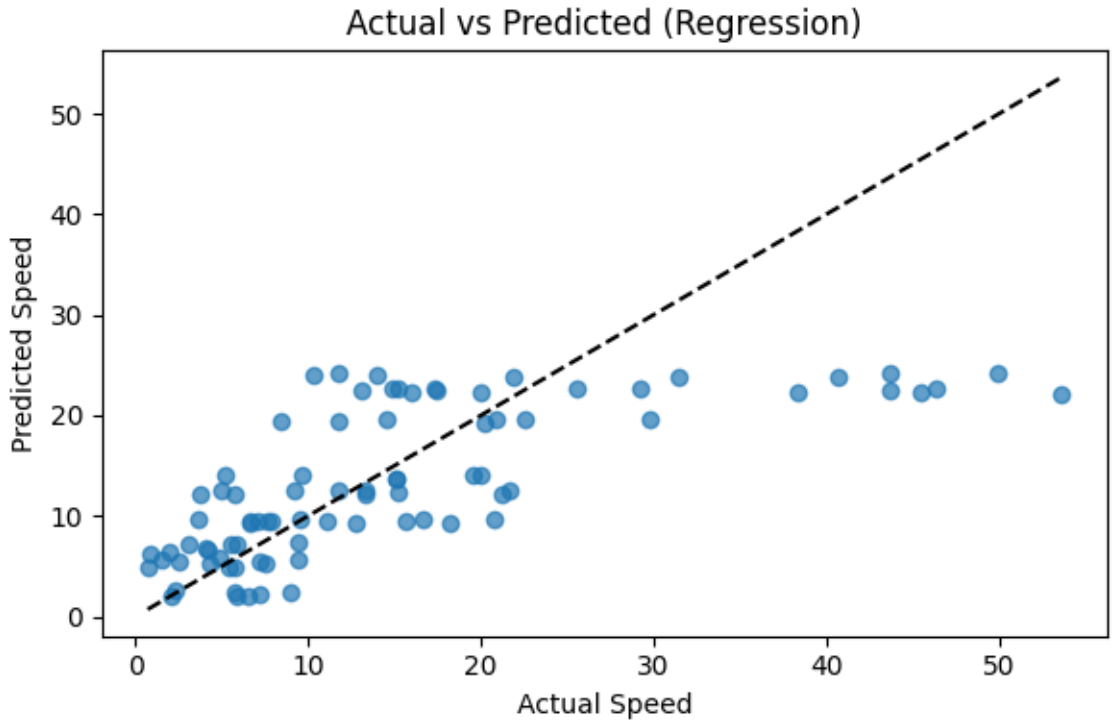

Traffic Flow and Sensor Analysis
10K
PeakHourVehicleCount
119.33
Max of vehicle_speed
28.97
Average of vehicle_speed
26K
Sum of vehicle_count
Street_Name
Agricultural Rd - Menoufia NU (Alexandria bound)
Agricultural Rd - Menoufia NU (Cairo bound)
Overpass Ramp - Menoufia NU (bridge)
congestion_level
2K (7.51%)
8K (32.3%)
15K (60.18%)
congestion_level
Low
Medium
High
Traffic Flow and Vehicle Performance Analysis by Street Name
hour_of_day
Overpass Ramp - Menoufia NU (...
Agricultural Rd - Menoufia NU (A...
Agricultural Rd - Menoufia NU (C...



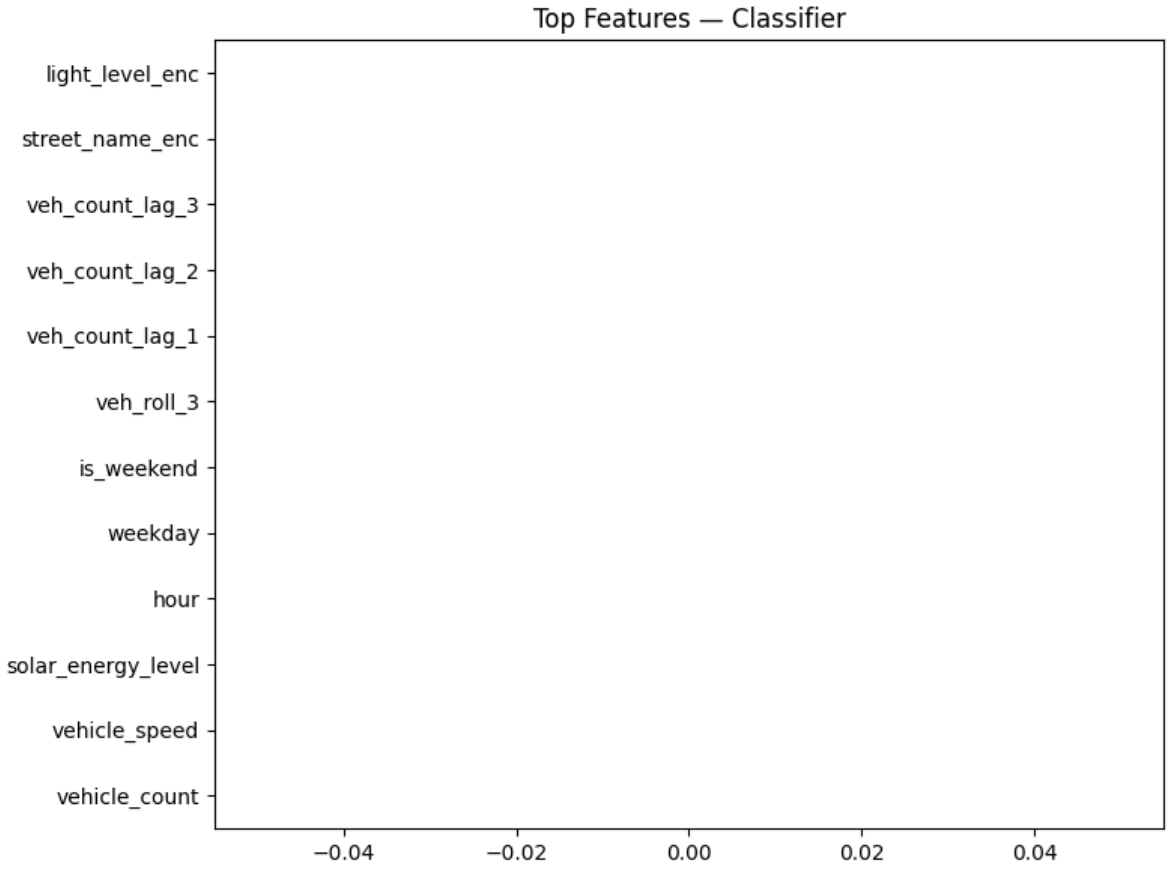
8. Visualizations

8.1 Congestion Regression Distribution





8.2 congestion classification distribution:



9. Tutorials (How to Run Everything)

Step-by-Step

Step 1: Run Docker Services

```
docker compose up -d
```

Step 2: Start MQTT → Kafka bridge

Step 3: Start IoT generator (Milestone 1)

Step 4: Run Spark Streaming job

Step 5: Verify Parquet folder

Step 6: SSIS Loads SQL Server & Archive Files

Step 7: Check SQL tables

Step 8: Test actuator (Lighting control)

10. Complete Technical Report

Sections Included:

- Introduction
- Problem definition
- Literature review
- System requirements
- Architectural design
- Data flow diagrams
- Database design
- Streaming pipeline
- Implementation
- Experiment & results
- Challenges
- Conclusion & future work

Traffic Prediction Project – Integrated Report

2. Project Objectives

The main objectives of the Traffic Prediction Project are:

4. **Traffic Congestion Prediction**

    a. Develop a machine learning model to predict traffic congestion levels based on historical and real-time traffic data.
5. **Real-time Data Integration**
    a. Build a data pipeline that ingests traffic data in real time using Kafka and processes it with Spark for timely predictions.
6. **Feature-based Analysis**
    a. Identify key factors affecting traffic congestion (e.g., average speed, hour of day, day of week, weekend indicators) to improve model accuracy.
7. **Decision Support for Traffic Management**
    a. Provide actionable insights for city traffic authorities to optimize traffic flow and reduce congestion during peak hours.
8. **Visualization & Monitoring**
    a. Implement visualizations and dashboards to monitor traffic trends, congestion hotspots, and model predictions.

3. Tools & Technologies

- **Python**: Data processing and machine learning
- **Pandas, NumPy, Matplotlib, Seaborn**: Data handling and visualization
- **Scikit-learn**: Random Forest Classifier
- **Kafka & Spark**: Real-time data ingestion and streaming
- **Joblib**: Model serialization
- **Jupyter Notebook**: Code development and interactive visualization

4. Data Processing & Features

- **Key Features Used:**
    - `avg_speed`: Average speed of vehicles
    - `hour`: Hour of the day
    - `dayofweek`: Day of the week
    - `is_weekend`: Weekend indicator (True/False)
    - `count_lag_1`: Traffic count from previous timestamp
    - `count_roll_3`: Rolling average of traffic counts over 3 periods
- **Data Cleaning Steps:**
    - Remove non-ASCII characters
    - Standardize column names (replace spaces with underscores, lowercase)
    - Handle missing values and outliers

5. Model Development

- **Algorithm Used:** Random Forest Classifier
- **Development Steps:**
  - Split the data into training and testing sets
  - Train the Random Forest model using selected features
  - Evaluate the model's accuracy and performance
  - Analyze feature importance to identify key contributors to traffic congestion
- **Feature Importance Analysis:**
  - `avg_speed` and `hour` were the most influential features
  - Lag and rolling features helped capture temporal patterns

6. Data Pipeline

- **Ingestion:** Kafka topics stream real-time traffic data
- **Processing:** Spark handles large-scale feature computation and preprocessing
- **Storage:** Processed data saved to a database or CSV for modeling
- **Prediction:** The trained ML model predicts congestion levels in real-time

7. Visualizations

- Traffic patterns by hour and day of the week
- Feature importance graphs
- Rolling averages and congestion trend plots
- Dashboard mockups for traffic monitoring

8. Current Status

Completed:

- Data cleaning and preprocessing
- Feature engineering
- Random Forest model training and evaluation
- Integration with Kafka and Spark for streaming data

- Feature importance visualization

Next Steps:

- Deploy the model for real-time predictions
- Build interactive dashboards for city traffic authorities
- Fine-tune model parameters for higher accuracy

9. Challenges & Solutions

- **Missing/Inconsistent Data:** Addressed with lag and rolling average features
- **Real-time Integration:** Resolved with Kafka streaming + Spark processing
- **Peak Hour Prediction Accuracy:** Improved via feature engineering (`hour`, `dayofweek`, `is_weekend`)

10. Conclusion

The Traffic Prediction Project successfully builds a predictive model that uses historical and real-time data to forecast congestion. The model provides actionable insights and lays the foundation for city-wide traffic monitoring and management. Future improvements include real-time dashboards, scaling the data pipeline, and continuous model optimization.

11. Results

✓ Stable real-time ingestion (MQTT → Kafka)

✓ Accurate data processing & alerting

✓ Dynamic lighting control

✓ Parquet lake created successfully

✓ SQL Server tables updated continuously

✓ SSIS automation for ETL

✓ Architecture scalable for ML in Milestone 4

12. Future Work

- Predictive models (already implemented in Milestone 4)
- Interactive web dashboard (React/Django)
- Geospatial heatmap (Leaflet.js + OpenStreetMap)
- Edge deployment on Raspberry Pi

13. Conclusion

This project implements a **fully functional smart-city system**, supporting real-time ingestion, processing, alerting, and optimization using modern data engineering frameworks. The pipeline is robust, scalable, and ready for future enhancements.