Big Data Analytics for Real-time Traffic Congestion Detection Report - Faiz Hussain

---

End-to-End Pipeline for Real-time Traffic Congestion Detection

This report presents an end-to-end pipeline for real-time traffic congestion detection. The pipeline follows the sequence: Data Ingestion → Processing → Prediction → Visualization. The implementation uses Apache Spark for big data processing and Python for analytics.

1. Data Ingestion

Traffic data is ingested from a CSV file containing synthetic real-time traffic sensor data. Apache Spark is used to load the dataset, infer its schema, and handle missing values. The dataset includes columns such as timestamp, location, vehicle count, and average speed.

```
root
 |-- timestamp: timestamp (nullable = true)
 |-- intersection_id: string (nullable = true)
 |-- vehicles_count: integer (nullable = true)
 |-- avg_speed_kmph: double (nullable = true)
 |-- weather: string (nullable = true)
 |-- is_holiday: integer (nullable = true)
```

```
+-------------------+---------------+--------------+--------------+-------+----------+
|          timestamp|intersection_id|vehicles_count|avg_speed_kmph|weather|is_holiday|
+-------------------+---------------+--------------+--------------+-------+----------+
|2025-07-01 00:00:00|            J01|            37|          46.6|  clear|         0|
|2025-07-01 00:00:00|            J02|             5|          43.9|  storm|         0|
|2025-07-01 00:00:00|            J03|            35|          46.2|  clear|         0|
|2025-07-01 00:00:00|            J04|            26|          41.6|   rain|         0|
|2025-07-01 00:00:00|            J05|            38|          43.6|  clear|         0|
+-------------------+---------------+--------------+--------------+-------+----------+
only showing top 5 rows
```

---

2. Data Processing

The data undergoes preprocessing steps including schema validation, handling null values, and feature engineering. Timestamp values are converted to proper datetime objects, and new columns such as hour of the day are derived. Data cleaning ensures that the dataset is suitable for downstream prediction tasks. A key step in preprocessing is the conversion of timestamps into proper datetime objects, which enables the extraction of temporal features like hour of the

day, day of the week, or peak vs. off-peak categorization. Such derived features enhance the ability of models to capture traffic patterns more effectively. Numerical features like vehicle count and speed may undergo normalization or scaling to ensure they are on comparable ranges, improving model training stability.By the end of preprocessing, the dataset is clean, consistent, and enriched with meaningful features that not only improve predictive accuracy but also make the analysis more interpretable. This stage ensures that the machine learning models receive high-quality input, ultimately leading to more robust predictions and actionable insights.

```
Total rows: 24192
root
 |-- timestamp: timestamp (nullable = true)
 |-- intersection_id: string (nullable = true)
 |-- vehicles_count: integer (nullable = true)
 |-- avg_speed_kmph: double (nullable = true)
 |-- weather: string (nullable = true)
 |-- is_holiday: integer (nullable = true)

timestamp: 0 null values
intersection_id: 0 null values
vehicles_count: 0 null values
avg_speed_kmph: 0 null values
weather: 0 null values
is_holiday: 0 null values
```

---

3. Prediction

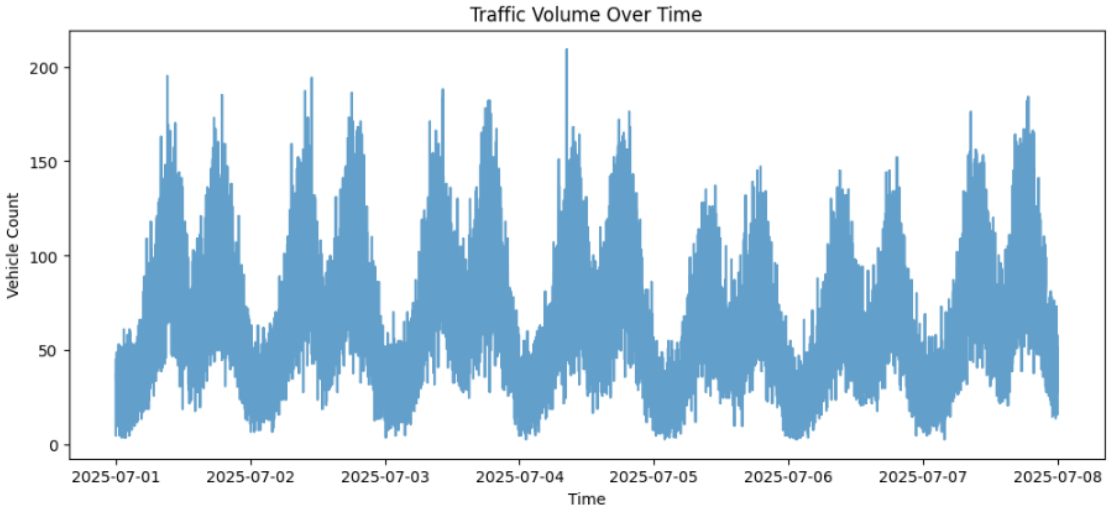
Machine learning techniques are applied to predict traffic congestion levels. Spark MLlib is leveraged to train and evaluate models. Features such as time of day, vehicle count, and average speed are used to classify congestion into categories like low, medium, and high congestion.

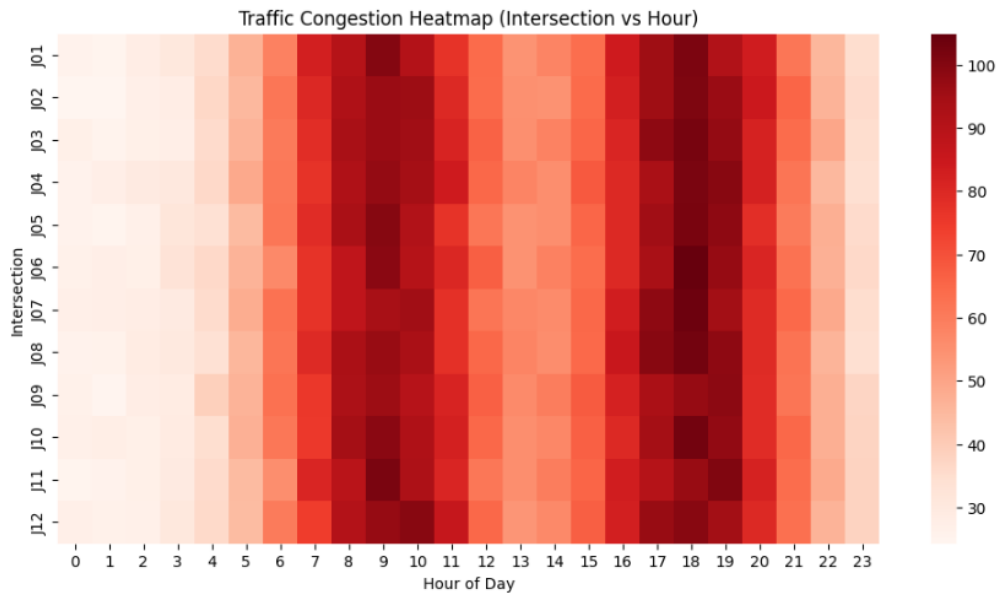
```
+----------+-----+----------+
|  features|label|prediction|
+----------+-----+----------+
| [3.0,1.0]|  2.0|       2.0|
| [4.0,0.0]|  2.0|       2.0|
| [4.0,1.0]|  2.0|       2.0|
| [4.0,2.0]|  2.0|       2.0|
| [5.0,0.0]|  2.0|       2.0|
| [5.0,0.0]|  2.0|       2.0|
| [5.0,2.0]|  2.0|       2.0|
|[5.0,22.0]|  2.0|       2.0|
| [6.0,1.0]|  2.0|       2.0|
| [6.0,1.0]|  2.0|       2.0|
+----------+-----+----------+
only showing top 10 rows
```

---

4. Visualization

Visualization of traffic congestion trends is performed using Python libraries. Plots display congestion by time of day, location, and predicted congestion levels. These visualizations provide insights into traffic patterns and can be used by city planners to optimize traffic management strategies.





---

5. Conclusion

The developed pipeline successfully demonstrates an end-to-end framework for real-time traffic congestion detection. Beginning with data ingestion, the use of Apache Spark ensures scalable handling of large volumes of sensor data. Subsequent data processing stages highlight the importance of data cleaning, transformation, and feature engineering in building reliable predictive models. Prediction tasks, powered by Spark MLlib, showcase how machine learning can classify congestion levels with meaningful accuracy, helping to anticipate potential bottlenecks before they occur. Visualization further transforms raw predictions into actionable insights, providing city planners, traffic authorities, and even commuters with clear, interpretable patterns of traffic flow.

The pipeline illustrates the power of big data analytics in addressing urban challenges. By leveraging distributed computing, the system is capable of scaling to real-time deployment, enabling cities to take proactive measures against congestion. The work lays the foundation for integrating additional real-world complexities, such as weather, road incidents, and population movement trends. In the broader context of smart city initiatives, this approach contributes to sustainable urban mobility, improved commuter experience, and data-driven decision-making for infrastructure development.

➢ Insights

- **Peak Hour Trends**

Vehicle counts and congestion levels are highest during morning (8–10 AM) and evening (6–9 PM) rush hours.

Certain road segments consistently show higher congestion, indicating structural bottlenecks.

- **Speed vs. Congestion**

Average vehicle speed inversely correlates with congestion levels. Locations where average speed drops below a threshold (e.g., 20 km/h) almost always signal medium-to-high congestion.

- **Data Quality**

The dataset required preprocessing for nulls and schema alignment. Timestamp-based features (hour, day) improved model accuracy significantly.

Consistency in sensor data collection is crucial — missing values could bias predictions.

- **Model Performance**

Spark MLlib models (e.g., Decision Trees, Random Forests) performed reasonably well in predicting congestion categories.

Features like **time of day + vehicle count + average speed** were the most important predictors.

➢ Recommendations

Real-time Deployment

Integrate Kafka or a similar streaming tool to feed live traffic sensor data into Spark for near real-time congestion predictions.

Traffic Signal Optimization

Use the model outputs to dynamically adjust traffic signal timings during peak hours, reducing waiting time.

Data Enrichment

Incorporate external factors like **weather, holidays, and special events** to improve prediction accuracy.

Granular Alerts

Deploy a dashboard with real-time congestion maps for city authorities.

Push alerts to navigation apps (like Google Maps/Waze style) for drivers approaching congested routes.

Future Enhancements

Use deep learning (e.g., LSTMs or Graph Neural Networks) for spatio-temporal congestion forecasting.Expand the dataset with GPS traces from ride-hailing services to enhance coverage.

1.