

Principles of Big Data Management

COMP-SCI 5540 [Spring Semester 2025]

Project Title: Real-Time Traffic Analysis with Big Data

Group Members:

Name	Student-ID
1. Jahnavi Redrouthu	16361448
2. Raj Kumar Bandi	16361456
3. Maheedhar Venigalla	16358619
4. Yeswanth Vallabhaneni	16359049

Abstract

The increasing complexity of urban transportation systems necessitates advanced methods for real-time traffic monitoring and decision support. This project presents the development of an integrated system that leverages Big Data technologies to collect, process, and visualize real-time traffic data with the goal of enhancing urban traffic management and planning. Centered on real-time vehicle speed data from major expressways and arterial roads, the system utilizes **Apache Kafka** for high-throughput data ingestion and **Spark Streaming** for scalable, low-latency processing. A robust ETL pipeline ensures data accuracy and transformation, while cloud infrastructure enables efficient storage and analysis. Machine learning models predict congestion patterns based on historical and real-time trends. An **interactive visualization dashboard** displays dynamic charts, real-time traffic trends, and a geo-mapped interface that reflect key metrics like congestion levels and average speed. It enables users to explore and analyze live traffic data through selectable attributes, supporting timely, data-driven decisions for urban mobility management.

1. Introduction

Business Problem Understanding

Urban mobility is becoming increasingly complex due to growing populations, limited infrastructure, and unpredictable traffic behaviors. Real-time traffic congestion management remains a critical challenge for modern metropolitan areas, especially those with dense vehicular flow and rapidly changing commuting patterns. Traditional traffic control systems are often reactive, offering limited agility in responding to dynamic urban conditions.

The Real-time Traffic Analysis project addresses this challenge by developing a data-driven visualization platform that continuously ingests, processes, and analyzes live traffic flow data. The system is designed to empower city planners, traffic control authorities, and commuters with timely, actionable insights that enhance traffic efficiency, reduce bottlenecks, and support long-term infrastructure decisions.

The project utilizes real-time traffic data streams sourced from road-based sensors and monitoring systems. These data feeds include key attributes such as speed, travel time, road segment identifiers, and timestamps, providing high-resolution visibility into traffic dynamics. By integrating real-time processing with intuitive, interactive visualizations—such as congestion timelines, speed-based charts, and geo-mapped traffic overlays—the system enables proactive decision-making in the face of complex urban mobility challenges.

Data Mining

The system captures real-time traffic data from public infrastructure feeds, collecting key attributes such as speed, travel time, road segment ID, and timestamps. This data is ingested through Apache Kafka, which ensures scalable, fault-tolerant data streaming.

Apache Spark Streaming processes this incoming data in near real-time, filtering and transforming it into structured formats suitable for analysis. The processed data is then sent to an interactive dashboard that visualizes key traffic metrics using dynamic charts and geo-mapped points.

This real-time pipeline enables users to explore traffic conditions, identify congestion trends, and interactively analyze data based on selected attributes—supporting quicker decision-making and improved situational awareness.

a. Project Scope

This project focuses on developing a real-time, data-driven traffic analysis system that continuously collects traffic data from public APIs. The system utilizes Apache Kafka for real-time data ingestion and Apache Spark Streaming for scalable data transformation and analysis. The insights are visualized through an interactive dashboard built with Flask, offering users an intuitive interface to monitor traffic conditions and key metrics in real time. The project is intended to support urban traffic management by providing timely visibility into current congestion levels, average speeds, and traffic volume across road segments.

Tasks:

- Set up Apache Kafka pipelines to stream live traffic data from APIs.
- Process incoming data using Spark Streaming to clean and structure it.
- Visualize key traffic metrics (speed, congestion, top sources) using dynamic charts and a real-time map in a Flask dashboard.
- Test and optimize system performance for responsiveness and usability.
- Document the full development and deployment process.

Deliverables:

- A web-based interactive dashboard displaying live traffic metrics (bar, pie, line charts, and map).
- A real-time pipeline for continuous ingestion and processing of traffic data.
- A responsive frontend UI built with HTML, CSS, and JavaScript.
- Documentation covering system architecture, usage, and testing.

Resources:

- **Technologies:** Apache Kafka, Apache Spark Streaming, Flask, Python, JavaScript (Chart.js, Leaflet)
- **Optional Extensions:** Future integration with ML models for predictive insights
- **Note:** The current project does not include machine learning or anomaly detection.

Timeline:

- **Phase 1: Planning & Design (March 28 – March 29, 2025)**
Define project goals, identify key data attributes, outline system architecture, and finalize tech stack (Kafka, Spark Streaming, Flask, Chart.js).
- **Phase 2: Data Collection & Integration (March 30 – April 1, 2025)**
Set up API access for live traffic data. Begin ingestion pipeline with Apache Kafka and validate real-time data flow.

- **Phase 3: Stream Processing Setup (April 2 – April 4, 2025)**
Implement Apache Spark Streaming to process incoming Kafka messages, perform basic cleaning and transformation.
- **Phase 4: Dashboard Development (April 5 – April 7, 2025)**
Build the interactive frontend using Flask, HTML, CSS, and JavaScript. Set up charts and dynamic visual elements.
- **Phase 5: Visualization Logic & KPI Metrics (April 8 – April 10, 2025)**
Implement features such as column-based filtering, real-time chart updates, traffic summaries, and the live map.
- **Phase 6: Testing & Optimization (April 11 – April 14, 2025)**
Run functional and integration tests, optimize data handling, improve UI responsiveness, and debug any issues.
- **Phase 7: Documentation & Final Review (April 15 – April 16, 2025)**
Write user and technical documentation, finalize system diagrams, and prepare for delivery.
- **Phase 8: Final Submission (April 17 – April 23, 2025)**
Package the project, submit deliverables, and present key outcomes and insights.

b. System Requirement Specifications (SRS)

- Functional Requirements

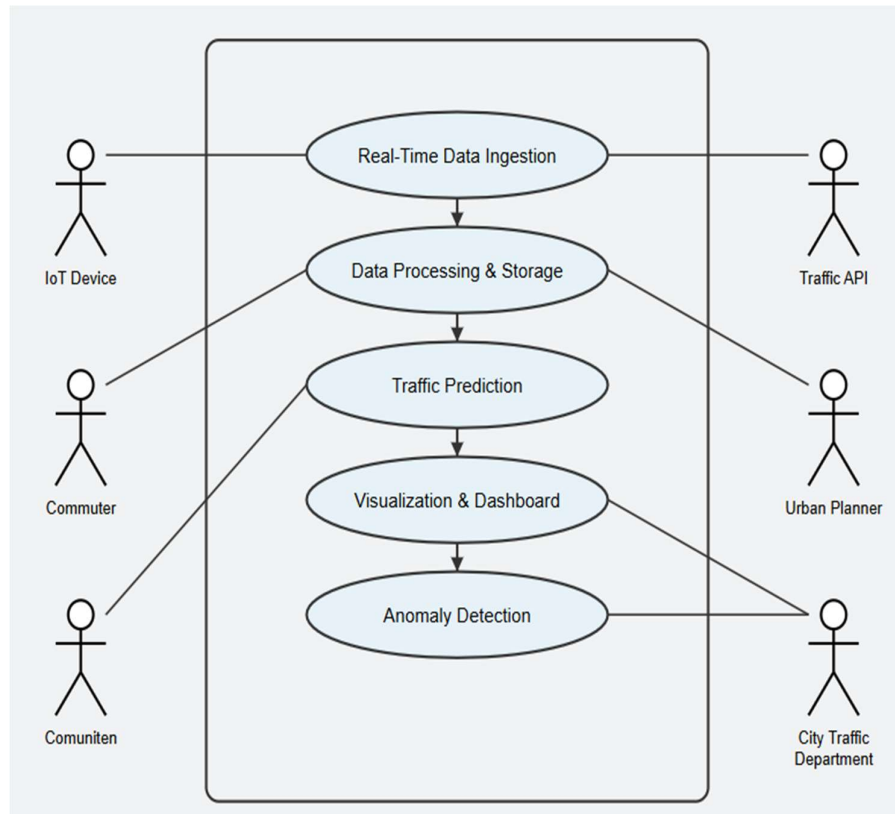
- **Real-time Data Ingestion:** Continuous collection of live traffic data from public APIs using Apache Kafka.
- **Stream Processing:** Real-time transformation and analysis of traffic data using Spark Streaming to detect patterns and anomalies.
- **Interactive Visualization:** Dashboard with dynamic charts and maps reflecting real-time traffic insights.

- Non-functional Requirements

- **Scalability:** Modular design to allow integration with additional data sources.
- **Fault Tolerance:** Use of distributed systems to ensure data delivery and system uptime.
- **Security:** Secure APIs and encrypted data transmission.

c. Use-Case Diagram

A use-case diagram will be provided showing user interactions with the system, including data sources (IoT devices, APIs), processing modules, and visualization components.



2. Data Engineering

a. ETL Process (Explanation):

Extract Phase

The extraction phase involves collecting raw traffic data from multiple sources:

- **Data Sources:** Traffic sensors, camera feeds, public APIs, IoT devices, GPS sensors, and social media platforms.
- **Data Elements:** Average vehicle speeds, road segment IDs, travel direction, and timestamps.
- **Technology:** Apache Kafka serves as the primary ingestion mechanism, creating a scalable, fault-tolerant data collection system.
- **Process:** Data is continuously streamed from sources into Kafka topics, enabling reliable, high-throughput data capture.

Transform Phase

Once extracted, the raw data undergoes several transformation processes:

- **Data Cleaning:** Removing incomplete or erroneous records, handling missing values, and filtering outliers.

- **Normalization:** Standardize the formats of incoming data to maintain uniformity. For example, ensure that timestamps are in the same format across different sources.
- **Feature Extraction:** Deriving additional attributes such as congestion levels, anomaly scores, and temporal patterns.
- **Aggregation:** Computing averages, maximums, minimums, and other statistical measures across time periods and road segments.
- **Technology:** Apache Spark Streaming enables real-time data transformation with low latency and high throughput.
- **Processing Logic:** Custom algorithms detect traffic patterns, identify anomalies, and prepare data for machine learning models.

Load Phase

The transformed data is then loaded into appropriate destinations:

- **Storage Systems:** Optimized databases for both real-time access and historical analysis.
- **Analytics Engines:** Loading processed data into systems that support predictive modeling.
- **Visualization Components:** Preparing and structuring data for the interactive dashboard.
- **Data Models:** Organizing information to support both immediate operational needs and long-term analytical requirements.

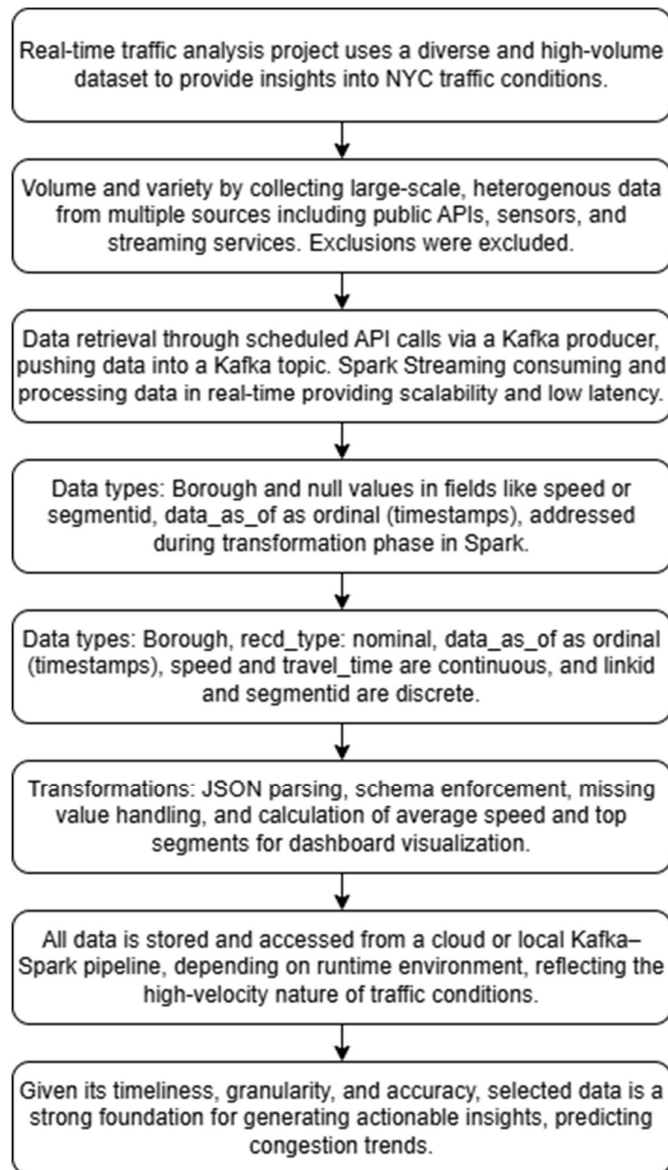
ETL Pipeline Architecture

The complete ETL pipeline is designed for:

- **Scalability:** Horizontal scaling to accommodate increasing data volumes.
- **Low Latency:** Near real-time processing to support immediate decision-making.
- **Reliability:** Fault tolerance through distributed processing and data replication.
- **Flexibility:** Adaptability to incorporate new data sources or analytical requirements.



b. Data Management



This real-time traffic analysis project utilizes a diverse and high-volume dataset to provide insights into traffic conditions in New York City.

The dataset focuses on real-time traffic data, including attributes such as speed, travel time, road segment IDs, borough, timestamps, and geographic coordinates. These variables are essential for analyzing congestion, identifying traffic bottlenecks, and providing real-time visualizations on a dashboard.

The project embraces volume and variety by collecting large-scale, heterogeneous data from multiple sources including public traffic APIs, sensors, and streaming services. This includes

both traditional structured data like segment IDs and timestamps, and non-traditional data such as geographic polyline encodings and congestion levels inferred from speed metrics.

The selected dataset from NYC's open data API is preferred due to its open availability, real-time refresh rate, consistent formatting, and relevance to the project's geographical scope. Other datasets were excluded mainly due to access restrictions, outdatedness, or lack of necessary attributes like speed or precise geolocation.

The data is retrieved using scheduled API calls via a Kafka producer, which pushes data into a Kafka topic. Spark Streaming is used downstream to consume, parse, and process this data in real-time, providing scalable and low-latency transformations.

Known issues include occasional missing or null values in fields like speed or segmentid, and inconsistencies in location data. These are addressed through basic cleaning techniques such as null filtering and outlier removal during the transformation phase in Spark.

Data types vary across features: borough and record_type are nominal, data_as_of is ordinal (timestamps), speed and travel_time are continuous, and fields like linkid and segmentid are discrete identifiers.

Transformations include JSON parsing, schema enforcement, missing value handling, and calculation of average speed and top segments. Aggregations and visual mappings are also applied for dashboard visualization.

All data is stored and accessed from a cloud or local Kafka-Spark pipeline, depending on the runtime environment, enabling real-time analysis and dashboard updates.

Data is refreshed every 10 seconds, reflecting the high-velocity nature of traffic conditions. This ensures that stakeholders can make timely and informed decisions.

Given its timeliness, granularity, and accuracy, the selected data is a strong foundation for generating actionable insights, predicting congestion trends, and supporting informed business decisions such as route optimization and urban planning.

3. Data Analytics and Modeling

- Descriptive Analytics

The system computes and displays aggregated metrics such as total traffic records, average congestion (based on speed), and the most congested areas by borough or segment. These summaries are visualized using:

KPI cards (e.g., total reports, average speed),

Bar and pie charts (distribution across selected attributes),

Line charts (temporal trends).

- Diagnostic Analytics

Interactive dashboards help users explore why certain traffic conditions occur. This is achieved through:

Drill-down capabilities by borough, link ID, or segment ID,

Analysis of attribute correlations (e.g., speed vs. travel_time),

Map visualizations showing traffic density and average speeds by location.

- Predictive Analytics (Framework Ready)

Although predictive modeling is not yet active, the data pipeline is designed to support real-time machine learning model integration. Forecasting future congestion trends using time-series analysis (e.g., ARIMA, LSTM) can be easily added to the current architecture.

- Modeling Techniques Applied

- Statistical Techniques

Aggregations like mean speed and frequency counts per segment.

Real-time calculations of averages and distributions.

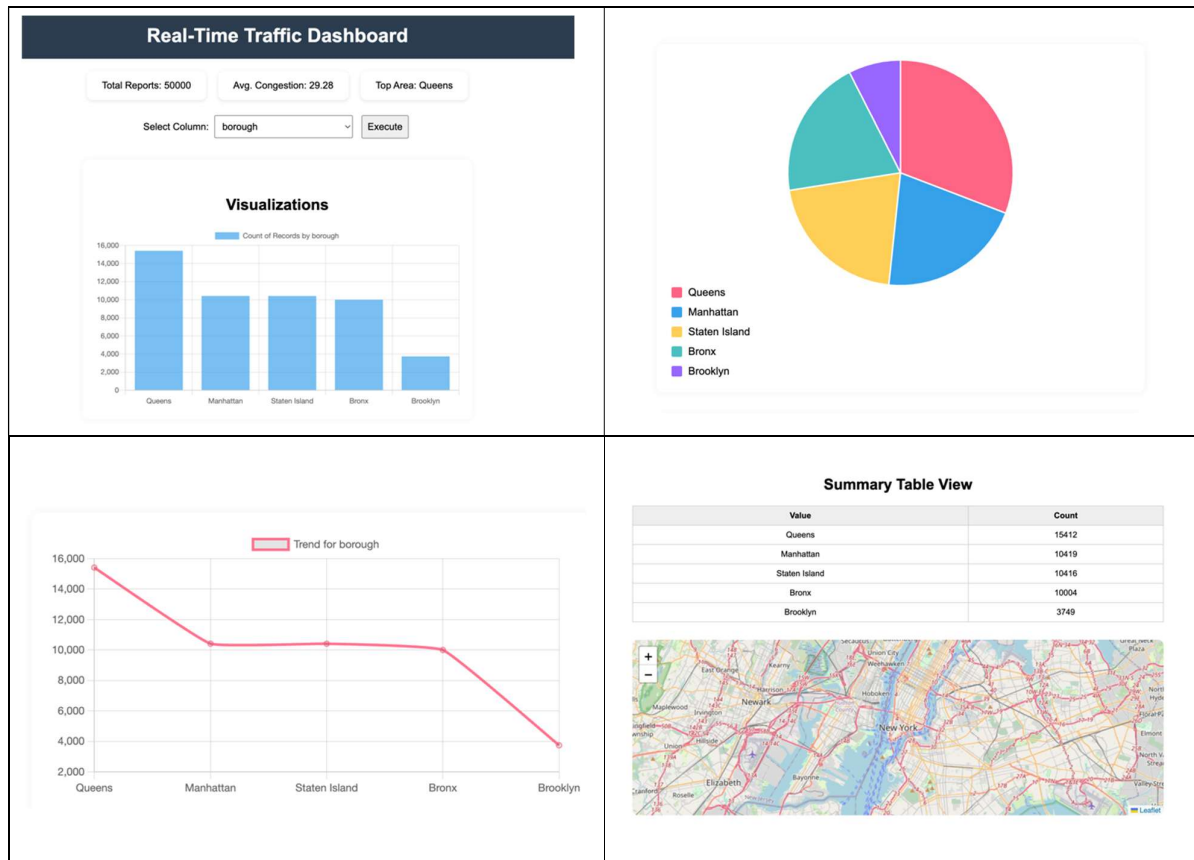
- Machine Learning Readiness

Spark Streaming allows scalable processing for future ML model training and inference.

Potential for anomaly detection (e.g., unusual speed drops) and clustering (e.g., segmenting traffic zones based on congestion patterns).

Layer	Tool / Tech Used	Description
Ingestion	Kafka Producer (kafka_producer.py)	Streams traffic JSON data from NYC Open Data API every 10s
Processing	Spark Streaming (spark_streaming.py)	Parses, transforms, and filters the data for real-time use
Frontend	Flask + JavaScript + Chart.js (index.html, script.js)	Displays data visuals and controls for filtering
Storage	In-memory Kafka topic; extensible to HDFS/S3	Supports fast, fault-tolerant streaming
Visualization	KPI Cards, Bar, Pie, and Line Charts	Interactive analytics rendered on dashboard

4. Data Visualization



5. Code

Link: <https://github.com/Jahnavi-Redrouthu/Real-Time-Traffic-Analysis-using-BIGDATA>

kafka_producer.py

```
import json
import time
import requests
from kafka import KafkaProducer
from kafka.errors import KafkaError
import signal
import sys
API_URL = "https://data.cityofnewyork.us/resource/i4gi-tjb9.json"
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
```

```

value_serializer=lambda v: json.dumps(v).encode('utf-8'),
retries=5,
acks='all',
request_timeout_ms=60000, # Increase the request timeout
max_block_ms=60000        # Increase the max block timeout for
metadata
)

```

Graceful shutdown handling

```
def signal_handler(sig, frame):
```

```

    print("Gracefully shutting down...")
    producer.flush()
    producer.close()
    sys.exit(0)

```

```

signal.signal(signal.SIGINT, signal_handler)
signal.signal(signal.SIGTERM, signal_handler)

```

```
while True:
```

```

    try:
        response = requests.get(API_URL)
        if response.status_code == 200:
            data = response.json()
            for record in data:
                try:
                    producer.send("traffic-topic",
value=record).get(timeout=10)
                except KafkaError as e:
                    print(f"Error sending record to Kafka: {e}")
                    print("Sent batch to Kafka.")
            else:
                print(f"Error fetching data: {response.status_code}")

            time.sleep(10) # Wait before next batch

    except Exception as e:
        print(f"An error occurred: {e}")
        time.sleep(10) # Wait before retrying

```

spark_straming.py

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import *

# Build Spark session with Kafka JARs included
spark = SparkSession.builder \
    .appName("TrafficDataProcessor") \
    .config("spark.jars",
            "/Users/jahnaviredrouthu/spark-sql-kafka-0-10_2.12-3.5.5.jar,"
            "/Users/jahnaviredrouthu/kafka-clients-3.5.5.jar,"
            "/Users/jahnaviredrouthu/commons-pool2-2.11.1.jar") \
    .getOrCreate()

# Define schema for the incoming data
schema = StructType([
    StructField("borough", StringType(), True),
    StructField("speed", DoubleType(), True),
    StructField("travel_time", DoubleType(), True),
    StructField("data_as_of", StringType(), True),
    StructField("linkid", StringType(), True),
    StructField("segmentid", StringType(), True)
])

# Read data from Kafka
df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "traffic-topic") \
    .load()

# Parse the JSON data from Kafka
traffic_df=df.selectExpr("CAST(value AS STRING
)").select(from_json(col("value"), schema).alias("data")).select("data.*")
# Output to console (for testing purposes)
query = traffic_df.writeStream \
    .format("console") \
    .start()
query.awaitTermination()

```

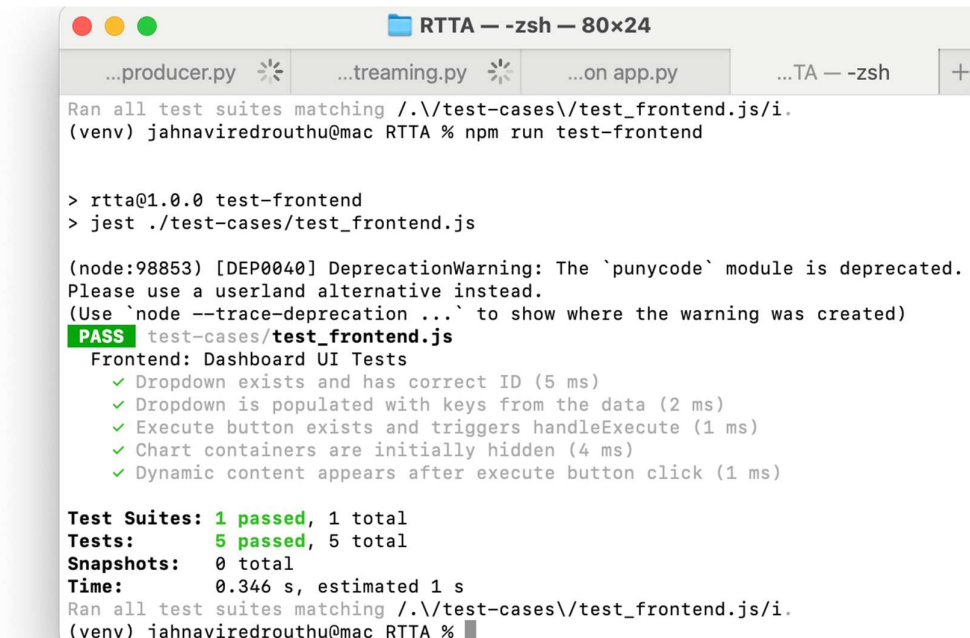
app.py

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("index.html")
if __name__ == '__main__':
    app.run(debug=True)
```

6. Test Cases

6.1.1 Frontend: Dashboard UI Tests

1. **Dropdown exists and has correct ID**
 - o Verify that the dropdown element exists and has the correct ID.
2. **Dropdown is populated with keys from the data**
 - o Ensure the dropdown is populated with keys from the mock data.
3. **Execute button exists and triggers handleExecute**
 - o Verify that the execute button exists and triggers the handleExecute function when clicked.
4. **Chart containers are initially hidden**
 - o Ensure that chart containers are initially hidden on page load.
5. **Dynamic content appears after execute button click**
 - o Verify that dynamic content (e.g., charts) appears after the execute button is clicked.



```
RTTA — -zsh — 80x24
...producer.py ...treaming.py ...on app.py ...TA — -zsh +
Ran all test suites matching ./test-cases/test_frontend.js/i.
(venv) jahnnaviredrouthu@mac RTTA % npm run test-frontend

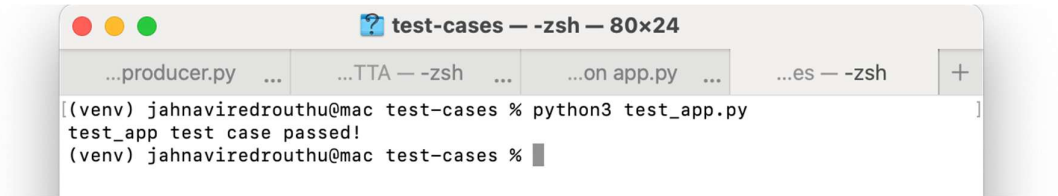
> rtta@1.0.0 test-frontend
> jest ./test-cases/test_frontend.js

(node:98853) [DEP0040] DeprecationWarning: The `punycode` module is deprecated.
Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS test-cases/test_frontend.js
  Frontend: Dashboard UI Tests
    ✓ Dropdown exists and has correct ID (5 ms)
    ✓ Dropdown is populated with keys from the data (2 ms)
    ✓ Execute button exists and triggers handleExecute (1 ms)
    ✓ Chart containers are initially hidden (4 ms)
    ✓ Dynamic content appears after execute button click (1 ms)

Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 0.346 s, estimated 1 s
Ran all test suites matching ./test-cases/test_frontend.js/i.
(venv) jahnnaviredrouthu@mac RTTA %
```

6.1.2 Backend Testing (Flask Route)

Index Route: Verify that the index route (/) returns a status code of 200 and contains the text "Real-Time Traffic Dashboard."



```
test-cases — -zsh — 80x24
...producer.py ... ...TTA — -zsh ... ...on app.py ... ...es — -zsh +
[(venv) jahnaviredrouthu@mac test-cases % python3 test_app.py
test_app test case passed!
(venv) jahnaviredrouthu@mac test-cases % ]
```

6.1.3 Kafka Producer Testing

Kafka Message Production: Verify that the Kafka producer sends JSON messages to the Kafka topic, which can be observed using Kafka console consumers.

7. References

- [1] Apache Software Foundation. (n.d.). *Apache Kafka Documentation*. Retrieved from <https://kafka.apache.org/documentation/>
- [2] Apache Spark. (n.d.). *Structured Streaming Programming Guide*. Retrieved from <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>
- [3] Zheng, C., Fan, X., Wang, C., & Qi, J. (2017). *Traffic flow prediction with big data: A deep learning approach*. arXiv preprint arXiv:1709.02824. Retrieved from <https://arxiv.org/abs/1709.02824>
- [4] Plotly Technologies Inc. (n.d.). *Plotly.js – The open source JavaScript graphing library*. Retrieved from <https://plotly.com/javascript/>
- [5] NYC Open Data. (n.d.). *Real-Time Traffic Speed Volume (i4gi-tjb9)*. Retrieved from <https://data.cityofnewyork.us/Transportation/Real-Time-Traffic-Speed-Volume/i4gi-tjb9>
- [6] Grinberg, M. (n.d.). *Flask Documentation*. Pallets Projects. Retrieved from <https://flask.palletsprojects.com/en/latest/>