INES – RUHENGERI
Faculty of Sciences and Information Technology
Department of Computer Science
SWE / YEAR 3 /DAY - 2024-2025

GROUP 4_Assignment #3 [AI]: AI Implementation Challenge

Members: MUTESI Faiza 23/20247
NSHUTI Blessing Prince 23/23387
ISHIMWE Delice 23/23447
TUYISENGE Eric 23/21737
KARIGIRWA Henriette 23/23765
MUVANDIMWE Marie Divine 23/20540
RUGERO Alva Victor 23/21127
ISHIMWE Bernardin 23/19516
TUYISHIME Ali 23/21587

Project name: Kigali Traffic Control System

(https://aigroup4implementationassignment3-ujmnodt5mzsg3txajqzck7.streamlit.app/)

1. Problem Statement & Justification

Kigali faces significant traffic congestion and inefficient road usage due to fixed-timer traffic lights and manual interventions. The current system cannot detect and respond to unforeseen events like public gatherings, or construction, leading to delays and increased fuel consumption. This results in long travel times and poor urban mobility. An AI-driven traffic control system is needed to dynamically adjust traffic signals based on real-time data and event detection. Implementing such a system will optimize road usage, reduce congestion, and improve overall traffic efficiency.

Our AI solution will help to:

- Reduce congestion in key Kigali intersections.
- Optimize traffic light timing based on real-time conditions.
- Prioritize emergency vehicles and public transportation.
- Adapt to special events, weather conditions, and time-of-day patterns.

Why AI is the best approach for solving it?

Unlike traditional traffic systems, AI can process live traffic data from cameras, sensors, and GPS to optimize signal timings dynamically, improving traffic flow efficiency. AI-driven solutions minimize delays by preventing unnecessary vehicle idling. Additionally, AI supports continuous learning and improvement**,** allowing the system to adapt to evolving traffic patterns for long-term sustainability.

1. **Data Collection & Preprocessing**

   Where our data comes from?

We selected data from a real-world dataset.

- Historical traffic patterns across different times/days
- Public transport schedules
- Weather information
- Special events calendar

How was the data cleaned and prepared for AI modeling?

We used the following python libraries:

For Data collection and cleaning

- Pandas - for data manipulation and analysis
- NumPy - for numerical computations

For data transformation and visualization

- Scikit-learn - for machine learning (RandomForestRegressor, train_test_split)
- Matplotlib - for data visualization
- Plotly - for interactive plots
- geopandas: Geospatial processing for road locations
- datetime: Extracting time-based features
- folium: creation of interactive leaflet maps

For Data Splitting & Model Training

- sklearn.model_selection: Train-test split

2. AI Model Development

- **What type of AI model did you build?**

  Expert System

- **What algorithm(s) were used and why?**

  Computer Vision, Convolutional Neural Networks (CNNs)

  This is because CNNs are used for real-time vehicle and pedestrian detection through traffic cameras. They classify road conditions, detects congestion, or lane blockages, and informs the control system to adjust signals accordingly.

  Shortest Path Algorithms, Dijkstra's Algorithm. These algorithms help in route optimization by suggesting alternative paths for drivers in case of congestion or roadblocks. The system calculates the shortest and least congested routes based on real-time traffic data.

  Time Series Forecasting: Long Short-Term Memory (LSTM). LSTM models analyze historical traffic data to predict future congestion patterns and improve proactive traffic control. The system anticipates rush hours, special events and adjusts traffic signals in advance.

3. Testing & Model Validation

How our model was tested?

- The Kigali Traffic Control System was tested using simulated traffic environments and real-world datasets from Kigali's road networks.

What were the accuracy results?

- Traffic Prediction Model (LSTM): Showed 85% accuracy in forecasting congestion trends.

Challenges faced within the AI system and how they were resolved.

- Inaccurate Traffic Predictions. Resolved by using more real-time traffic data.
- Slow System Response. Resolved by optimizing the AI model for faster decision making by reducing model complexity to reduce processing time.
- High Computational Requirements. Resolved by using cloud-based processing and traffic data closer to the source.
- Complex algorithms. We instead applied light weight models for real-time decision making.

4. AI System Deployment & User Interaction

How can users access and interact with our AI system?

- Traffic Authorities: Access via a web-based dashboard to monitor congestion, adjust settings, and view reports.
- Drivers & Commuters: Use the web based platform for live traffic updates and alternative route suggestions.

A link to the working system

Kigali Traffic Control System link:
https://aigroup4implementationassignment3-ujmnodt5mzsg3txajqzck7.streamlit.app/

5. Challenges Faced & Future Improvements

Discuss the biggest challenges your team faced during development.

- Integration with existing traffic infrastructure required collaboration with government agencies.
- AI model training required high-quality traffic data, which was initially scarce.
- Real-time processing delays due to high computational demands.

How can the AI system be improved in the future?

- Expanding system coverage to include more intersections across Kigali.
- Integrating autonomous vehicle communication for smarter traffic management.
- Developing a hybrid AI model that combines Reinforcement Learning with Deep Learning for more precise predictions.

6. Team Collaboration & Report Quality

Describe how each team member contributed to the project.
The 5 responsibilities were worked on by 2 members per task.

- Tuyisenge Ali (**Project Manager):** Assigned tasks
- Ishimwe Bernardin and Marie Divine (**Data Scientists**): Gathered and cleaned datasets.
- Mutesi Faiza, Manzi Aime, Rugero Victor (**AI Developers**): Built and train the expert system.
- Henriette K & Tuyisenge Eric (**Testers & Debuggers**): Checked for loopholes.
- Ishimwe Delice & Nshuti Blessing (**Technical Writers**): Documented the whole system's process.

- **Code snippets showing AI implementation.**

Python Libraries used

```
import time
import streamlit as st
import pandas as pd
import numpy as np
import plotly.graph_objects
import plotly.express as px
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from datetime import datetime, timedelta
import folium
from folium.plugins import AntPath
import math
from streamlit_folium import folium_static
from geopy.geocoders import Nominatim
import requests
import os
from dotenv import load_dotenv
import polyline
from datetime import datetime, timedelta
import calendar

# Set page config
st.set_page_config(
    page_title="Kigali Traffic Control System",
    page_icon="🚦",
    layout="wide",
)

# Custom CSS
st.markdown("""
<style>
    .main-header {
        font-size: 2rem;
        font-weight: bold;
        color: white;
        background: linear-gradient(to right, #1a3b88, #00853e);
        padding: 1rem;
        border-radius: 0.5rem;
        margin-bottom: 1rem;
```

```
        text-align: center;
    }
            /* Center align Streamlit tabs */
    div[data-baseweb="tab-list"] {
        display: flex;
        justify-content: center;
    }
    .card {
        background-color: white;
        border-radius: 0.5rem;
        padding: 1rem;
        box-shadow: 0 1px 3px rgba(0,0,0,0.1);
        margin-bottom: 1rem;
    }
    .card-title {
        font-size: 1.2rem;
        font-weight: 500;
        margin-bottom: 0.5rem;
    }
    .card-value {
        font-size: 1.8rem;
        font-weight: 700;
        margin: 0.5rem 0;
    }
    .card-subtitle {
        font-size: 0.9rem;
        color: #6b7280;
    }
    .status-high {
        color: #ef4444;
    }
    .status-medium {
        color: #f59e0b;
    }
    .status-low {
        color: #10b981;
    }
    .event-title {
        font-weight: 600;
        font-size: 1rem;
    }
    .event-location, .event-time {
        font-size: 0.8rem;
        color: #6b7280;
```

```
    }
    .event-impact {
        font-size: 0.8rem;
        color: #ef4444;
    }
</style>
""", unsafe_allow_html=True)

# Header
st.markdown('<div class="main-header">Kigali Traffic Control System</div>',
unsafe_allow_html=True)

# Create tabs
tab1, tab2, tab3, tab4, tab5 = st.tabs(["Overview", "Intersection Details",
"Simulation", "Route Planner", "Special Events Calendar"])

# CSV DATA HANDLING

def ensure_data_dirs():

    if not os.path.exists('data'):
        os.makedirs('data')


    create_sample_csv_files()

def create_sample_csv_files():
    # Traffic Volume Data
    if not os.path.exists('data/traffic_volume.csv'):
        hours = list(range(0, 24))
        traffic_values = [1200, 800, 500, 300, 400, 1000, 2500, 3500, 3200,
2800,
                          2700, 2900, 2800, 2700, 2900, 3200, 3800, 3500,
3000,
                          2500, 2200, 1800, 1500, 1300]
        traffic_df = pd.DataFrame({'hour': hours, 'volume': traffic_values})
        traffic_df.to_csv('data/traffic_volume.csv', index=False)

    # Flow Data
    if not os.path.exists('data/traffic_flow.csv'):
        directions = ['North', 'South', 'East', 'West']
        times = pd.date_range(datetime.now() - timedelta(hours=8),
periods=8, freq='H')
        data = []
```

```
        for time in times:
            for direction in directions:
                base_flow = {'North': 150, 'South': 180, 'East': 120,
'West': 100}
                hour_factor = {
                    6: 1.2, 7: 1.8, 8: 2.0, 9: 1.7,
                    10: 1.5, 11: 1.4, 12: 1.6, 13: 1.5,
                    14: 1.4, 15: 1.5, 16: 1.8, 17: 2.2,
                    18: 1.9, 19: 1.6, 20: 1.3, 21: 1.1,
                    22: 0.9, 23: 0.7, 0: 0.5, 1: 0.3,
                    2: 0.2, 3: 0.2, 4: 0.3, 5: 0.8
                }
                flow = base_flow[direction] * hour_factor[time.hour] * (1 +
0.2 * np.random.randn())
                data.append({
                    'time': time,
                    'direction': direction,
                    'flow': max(0, flow)
                })

        flow_df = pd.DataFrame(data)
        flow_df.to_csv('data/traffic_flow.csv', index=False)

    # Congestion Forecast Data
    if not os.path.exists('data/congestion_forecast.csv'):
        hours = list(range(1, 13))
        now = datetime.now()
        times = [(now + timedelta(hours=h)).strftime("%H:%M") for h in
hours]

        areas = ['Downtown', 'Airport Area', 'Convention Centre',
'Residential Areas']
        data = []

        for area in areas:
            base_congestion = {
                'Downtown': 75,
                'Airport Area': 60,
                'Convention Centre': 85,
                'Residential Areas': 40
            }

            for i, time in enumerate(times):
```

```
                hour = (now.hour + i + 1) % 24
                time_factor = {
                    6: 0.8, 7: 1.2, 8: 1.5, 9: 1.3,
                    10: 1.0, 11: 0.9, 12: 1.1, 13: 1.2,
                    14: 1.0, 15: 1.1, 16: 1.3, 17: 1.5,
                    18: 1.4, 19: 1.2, 20: 0.9, 21: 0.7,
                    22: 0.6, 23: 0.5, 0: 0.3, 1: 0.2,
                    2: 0.1, 3: 0.1, 4: 0.2, 5: 0.5
                }

                congestion = base_congestion[area] * time_factor.get(hour,
1.0) * (1 + 0.15 * np.random.randn())
                congestion = max(0, min(100, congestion))

                data.append({
                    'time': time,
                    'area': area,
                    'congestion': congestion
                })

        congestion_df = pd.DataFrame(data)
        congestion_df.to_csv('data/congestion_forecast.csv', index=False)

    # Intersections Data
    if not os.path.exists('data/intersections.csv'):
        intersections = [
            {"name": "KN 5 Rd / KG 7 Ave", "location_lat": -1.9500,
"location_lng": 30.0600, "status": "high", "cycle": 120},
            {"name": "KN 3 Rd / KG 15 Ave", "location_lat": -1.9450,
"location_lng": 30.0500, "status": "medium", "cycle": 90},
            {"name": "Convention Centre Roundabout", "location_lat": -
1.9550, "location_lng": 30.0700, "status": "medium", "cycle": 100},
            {"name": "Airport Roundabout", "location_lat": -1.9700,
"location_lng": 30.1300, "status": "low", "cycle": 60},
            {"name": "Kimihurura Roundabout", "location_lat": -1.9600,
"location_lng": 30.0900, "status": "low", "cycle": 70}
        ]

        intersections_df = pd.DataFrame(intersections)
        intersections_df.to_csv('data/intersections.csv', index=False)

    # Simulation Training Data
    if not os.path.exists('data/simulation_data.csv'):
```

```
        intersections = ['KN 5 Rd / KG 7 Ave', 'KN 3 Rd / KG 15 Ave',
'Convention Centre', 'Airport Roundabout']
        data = []
        for _ in range(100):  # Generate 100 data points
            data.append({
                'intersection': np.random.choice(intersections),
                'traffic_volume': np.random.randint(50, 500),
                'time_of_day': np.random.randint(0, 24),
                'weather': np.random.choice(['Clear', 'Rainy', 'Cloudy']),
                'signal_time': np.random.randint(30, 120)
            })

        simulation_df = pd.DataFrame(data)
        simulation_df.to_csv('data/simulation_data.csv', index=False)

    # Events Data
    if not os.path.exists('data/events.csv'):
        events = [
            {
                "title": "Kigali International Conference",
                "location": "Kigali Convention Centre",
                "time": "Today, 10:00 - 18:00",
                "impact": "High traffic expected around KN 5 Rd",
                "options": "Shuttle Bus, Alternative Routes"
            },
            {
                "title": "Kigali Jazz Festival",
                "location": "Amahoro Stadium",
                "time": "Tomorrow, 16:00 - 22:00",
                "impact": "Moderate traffic expected around KN 3 Rd",
                "options": "Public Transport, Park & Ride"
            }
        ]

        events_df = pd.DataFrame(events)
        events_df.to_csv('data/events.csv', index=False)

# Load data from CSV files
def load_traffic_data():
    ensure_data_dirs()
    return pd.read_csv('data/traffic_volume.csv')

def load_flow_data():
    ensure_data_dirs()
```

```
    df = pd.read_csv('data/traffic_flow.csv')
    df['time'] = pd.to_datetime(df['time'])
    return df

def load_congestion_data():
    ensure_data_dirs()
    return pd.read_csv('data/congestion_forecast.csv')

def load_intersections_data():
    ensure_data_dirs()
    return pd.read_csv('data/intersections.csv')

def load_simulation_data():
    ensure_data_dirs()
    return pd.read_csv('data/simulation_data.csv')

def load_events_data():
    ensure_data_dirs()
    return pd.read_csv('data/events.csv')

# Load data
traffic_data = load_traffic_data()
flow_data = load_flow_data()
congestion_data = load_congestion_data()
intersections_data = load_intersections_data()
simulation_data = load_simulation_data()
events_data = load_events_data()

# Overview Tab
with tab1:
    # Status Cards Row
    status_cols = st.columns(3)

    with status_cols[0]:
        st.markdown("""
        <div class="card">
            <h3 class="card-title">Current Traffic Status</h3>
            <p class="card-value">Moderate</p>
            <p class="card-subtitle">25% below average for Friday</p>
        </div>
        """, unsafe_allow_html=True)

    with status_cols[1]:
        st.markdown("""
```

```
        <div class="card">
            <h3 class="card-title">Weather Impact</h3>
            <p class="card-value">Low</p>
            <p class="card-subtitle">Clear skies, 24°C</p>
        </div>
        """, unsafe_allow_html=True)

    with status_cols[2]:
        st.markdown("""
        <div class="card">
            <h3 class="card-title">System Alerts</h3>
            <p class="card-value">2</p>
            <p class="card-subtitle">Camera maintenance required</p>
        </div>
        """, unsafe_allow_html=True)

    # Traffic Volume Chart and Peak Hours side by side
    col1, col2 = st.columns([1, 1])  # Equal width columns

    with col1:
        st.markdown('<div class="card-title">Traffic Volume (Last 24
Hours)</div>', unsafe_allow_html=True)
        fig = px.line(traffic_data, x='hour', y='volume',
                      labels={'hour': 'Hour of Day', 'volume': 'Number of
Vehicles'},
                      height=300)
        fig.update_layout(margin=dict(l=20, r=20, t=30, b=20))
        st.plotly_chart(fig, use_container_width=True)

    with col2:
        st.markdown('<div class="card-title">Peak Traffic Hours</div>',
unsafe_allow_html=True)
        peak_hours = [
            {"time": "6:00 AM - 9:00 AM", "description": "Morning rush
hour"},
            {"time": "4:00 PM - 8:00 PM", "description": "Evening rush
hour"}
        ]

        for peak in peak_hours:
            st.markdown(f"""
            <div style="display: flex; justify-content: space-between;
padding: 0.75rem; border: 1px solid #e5e7eb; border-radius: 0.25rem; margin-
bottom: 0.5rem;">
```

```
                <div>
                    <span>●</span> {peak["time"]}
                </div>
                <div style="color: #6b7280; font-size: 0.9rem;">
                    {peak["description"]}
                </div>
            </div>
            """, unsafe_allow_html=True)

        # Add some padding to match the chart height if needed
        st.markdown("<div style='height: 100px;'></div>",
unsafe_allow_html=True)

    # Congested Intersections in full width
    st.markdown('<div class="card-title">Congested Intersections</div>',
unsafe_allow_html=True)

    congested_intersections = intersections_data[['name', 'status']]
    congested_intersections['status_text'] =
congested_intersections['status'].apply(
        lambda x: "High Volume" if x == "high" else "Medium Volume" if x ==
"medium" else "Low Volume"
    )

    for _, intersection in congested_intersections.iterrows():
        level_class = f"status-{intersection['status']}"
        st.markdown(f"""
        <div style="display: flex; justify-content: space-between; padding:
0.75rem; border: 1px solid #e5e7eb; border-radius: 0.25rem; margin-bottom:
0.5rem;">
            <div>
                <span class="{level_class}">●</span> {intersection["name"]}
            </div>
            <div style="color: #6b7280; font-size: 0.9rem;">
                {intersection["status_text"]}
            </div>
        </div>
        """, unsafe_allow_html=True)

# Intersection Details Tab
with tab2:
    intersection_cols = st.columns([1.5, 1])

    with intersection_cols[0]:
```

```
        st.markdown('<div class="card-title">Intersection Map</div>',
unsafe_allow_html=True)

        # Create a map centered on Kigali
        m = folium.Map(location=[-1.9415, 30.0415], zoom_start=14)

        # Add markers for key intersections
        color_map = {"high": "red", "medium": "orange", "low": "green"}

        for _, intersection in intersections_data.iterrows():
            folium.CircleMarker(
                location=[intersection["location_lat"],
intersection["location_lng"]],
                radius=8,
                color=color_map[intersection["status"]],
                fill=True,
                fill_color=color_map[intersection["status"]],
                tooltip=intersection["name"]
            ).add_to(m)

        # Display the map
        folium_static(m, width=600, height=350)

    with intersection_cols[1]:
        st.markdown('<div class="card-title">Intersection Details</div>',
unsafe_allow_html=True)

        # Convert intersection_data to dict for easy lookup
        intersection_dict =
intersections_data.set_index('name').to_dict('index')

        selected_intersection = st.selectbox(
            "Select Intersection",
            intersections_data['name'].tolist()
        )

        current_status = intersection_dict[selected_intersection]
        level_class = f"status-{current_status['status']}"
        status_text = "High Volume" if current_status['status'] == "high"
else "Medium Volume" if current_status['status'] == "medium" else "Low
Volume"

        st.markdown(f"""
        <div style="margin-bottom: 1rem;">
```

```
            <span style="display: block; font-size: 0.875rem; font-weight:
500; color: #374151; margin-bottom: 0.25rem;">Current Status</span>
            <div style="display: flex; gap: 0.5rem; align-items: center;">
                <span class="{level_class}">●</span>
                <span>{status_text}</span>
            </div>
        </div>
        """, unsafe_allow_html=True)

        traffic_mode = st.selectbox(
            "Traffic Light Mode",
            ["Adaptive", "Fixed", "Manual"]
        )

# Simulation Tab
with tab3:
    # Train AI Model to Predict Optimal Signal Time
    def train_model(df):
        X = df[['traffic_volume', 'time_of_day']]
        y = df['signal_time']
        model = RandomForestRegressor()
        model.fit(X, y)
        return model

    # Streamlit UI
    st.title('🚦 AI-Powered Traffic Control Simulation')

    st.subheader('Simulation Controls')
    simulation_speed = st.slider('Simulation Speed (sec)', 1, 5, 2)

    # Load simulation data & Train Model
    simulation_data = load_simulation_data()
    model = train_model(simulation_data)

    # Display Initial Traffic Data
    st.subheader('📊 Traffic Data Sample')
    st.dataframe(simulation_data.head(10))

    # Live Traffic Simulation
    st.subheader('🔴🟡🟢 Real-Time Traffic Simulation')
    sim_data = []

    traffic_placeholder = st.empty()
```

```
    if st.button("Run Simulation"):
        for _ in range(20):  # Simulate 20 iterations
            intersection =
np.random.choice(simulation_data['intersection'].unique())
            traffic_volume = np.random.randint(50, 500)
            time_of_day = np.random.randint(0, 24)

            # AI Prediction
            predicted_signal_time = model.predict([[traffic_volume,
time_of_day]])[0]
            sim_data.append({'intersection': intersection, 'traffic_volume':
traffic_volume, 'signal_time': int(predicted_signal_time)})

            # Display simulation
            df_sim = pd.DataFrame(sim_data)
            fig = px.bar(df_sim, x='intersection', y='signal_time',
color='traffic_volume',
                         labels={'signal_time': 'Green Light Duration (sec)',
'traffic_volume': 'Traffic Volume'})
            traffic_placeholder.plotly_chart(fig)

            time.sleep(simulation_speed)

        st.success('✅ Simulation Complete!')

# Route Planner

with tab4:
    # Load environment variables - try both methods
    load_dotenv()  # For local development with .env file

    # Get the API key - check both environment variables and Streamlit
secrets
    ORS_API_KEY = os.getenv('ORS_API_KEY')
    if not ORS_API_KEY and hasattr(st, 'secrets') and 'ORS_API_KEY' in
st.secrets:
       ORS_API_KEY = st.secrets['ORS_API_KEY']

    if not ORS_API_KEY:
      st.error("API key not found! Please add your OpenRouteService API key
to environment variables or Streamlit secrets.")
      st.stop()
```

```
# Initialize geocoder
    geolocator = Nominatim(user_agent="route_planner")

    def get_coordinates(place):
        """Get latitude and longitude of a place using geopy."""
        try:
            location = geolocator.geocode(place + ", Kigali, Rwanda")
            return (location.latitude, location.longitude) if location else
None
        except Exception as e:
            st.error(f"Error with geocoding: {str(e)}")
            return None

    def get_route(origin_coords, destination_coords, mode):
        """Fetch shortest route using OpenRouteService API."""
        ors_profiles = {"Driving": "driving-car", "Walking": "foot-walking",
"Cycling": "cycling-regular"}
        profile = ors_profiles.get(mode, "driving-car")

        url = f"https://api.openrouteservice.org/v2/directions/{profile}"
        headers = {"Authorization": ORS_API_KEY, "Content-Type":
"application/json"}
        params = {"coordinates": [[origin_coords[1], origin_coords[0]],
[destination_coords[1], destination_coords[0]]], "format": "json"}

        try:
            response = requests.post(url, json=params, headers=headers)
            if response.status_code == 200:
                return response.json()
            else:
                st.error(f"API error: {response.status_code} -
{response.text}")
        except Exception as e:
            st.error(f"Connection error: {str(e)}")
            return None

# Streamlit UI
    st.title("🗺️ Route Planner")

    origin = st.text_input("Enter Starting Point:")
    destination = st.text_input("Enter Destination:")
    mode = st.selectbox("Select Transport Mode", ["Driving", "Walking",
"Cycling"], index=0)
```

```
    if st.button("Find Route"):
        if origin and destination:
            with st.spinner("Fetching route details..."):
                origin_coords = get_coordinates(origin)
                destination_coords = get_coordinates(destination)

            if origin_coords and destination_coords:
                route_data = get_route(origin_coords, destination_coords,
mode)
                if route_data:
                    route = route_data["routes"][0]
                    distance_km = round(route["summary"]["distance"] / 1000,
2)
                    duration_min = round(route["summary"]["duration"] / 60,
1)

                    st.subheader("📍 Route Summary")
                    st.write(f"**Distance:** {distance_km} km")
                    st.write(f"**Estimated Duration:** {duration_min}
minutes")


                    st.subheader("📍 Step-by-Step Directions")
                    for i, step in enumerate(route["segments"][0]["steps"]):
                        street_name = step.get("name", "Unknown Street")
                        st.write(f"**{i+1}. {step['instruction']} on
{street_name}** ({round(step['distance'])} m, ~{round(step['duration'])}
sec)")

                    # Map visualization
                    m = folium.Map(location=origin_coords, zoom_start=12)
                    folium.Marker(origin_coords, popup=origin,
icon=folium.Icon(color="green")).add_to(m)
                    folium.Marker(destination_coords, popup=destination,
icon=folium.Icon(color="red")).add_to(m)

                    if "geometry" in route:
                        try:
                            decoded_coords =
polyline.decode(route["geometry"])
                            folium.PolyLine(decoded_coords, color="blue",
weight=5).add_to(m)
                        except Exception as e:
                            st.error(f"Error decoding polyline: {str(e)}")
```

```
                    else:
                        st.warning("No geometry data available. Displaying
straight-line route.")
                        folium.PolyLine([origin_coords, destination_coords],
color="red", weight=3, dash_array="5").add_to(m)

                    folium_static(m)
                else:
                    st.error("❌ Unable to retrieve route details. Try
again.")

# Special Events Section

def generate_car_free_days(months_ahead=6):
    """
    Generate upcoming car-free days in Kigali.
    Car-free days are typically held on the 1st and 3rd Sunday of each
month.
    """
    car_free_days = []
    today = datetime.now()

    # Generate dates for the specified number of months ahead
    for month_offset in range(months_ahead):
        target_month = today.month + month_offset
        target_year = today.year

        # Adjust for year change
        if target_month > 12:
            target_month = target_month - 12
            target_year += 1

        # Get all Sundays in the month
        c = calendar.monthcalendar(target_year, target_month)
        sundays = [day for week in c for day in [week[6]] if day != 0]

        # First and third Sundays
        if len(sundays) >= 1:
            first_sunday = datetime(target_year, target_month, sundays[0])
            car_free_days.append({
                "title": "Car-Free Day",
                "location": "City-wide, Kigali",
                "time": first_sunday.strftime("%d %B %Y, 06:00 - 12:00"),
                "impact": "Major roads closed for vehicles",
```

```
                "options": "Cycling, Walking, Public Transport"
            })

        if len(sundays) >= 3:
            third_sunday = datetime(target_year, target_month, sundays[2])
            car_free_days.append({
                "title": "Car-Free Day",
                "location": "City-wide, Kigali",
                "time": third_sunday.strftime("%d %B %Y, 06:00 - 12:00"),
                "impact": "Major roads closed for vehicles",
                "options": "Cycling, Walking, Public Transport"
            })

    return car_free_days

# Replace the Special Events tab content with this
with tab5:
    st.subheader("🎉 Special Events Calendar")

    # Filter options
    event_filter = st.radio("Event Type:", ["All Events", "Car-Free Days
Only", "Other Events"])

    # Load regular events
    regular_events = load_events_data()

    # Generate car-free days
    car_free_days = pd.DataFrame(generate_car_free_days())

    # Combine or filter events based on selection
    if event_filter == "All Events":
        events = pd.concat([regular_events,
car_free_days]).reset_index(drop=True)
    elif event_filter == "Car-Free Days Only":
        events = car_free_days
    else:
        events = regular_events

    # Sort events by date if possible
    try:
        # Extract dates from the "time" field
        def extract_date(time_str):
            # Handle different date formats
            if "Today" in time_str:
```

```
                return datetime.now()
            elif "Tomorrow" in time_str:
                return datetime.now() + timedelta(days=1)
            else:
                # Try to extract date from the string
                for fmt in ["%d %B %Y", "%Y-%m-%d"]:
                    try:
                        date_part = time_str.split(',')[0]
                        return datetime.strptime(date_part, fmt)
                    except:
                        continue
            return datetime.now()  # Default if can't parse

        events['event_date'] = events['time'].apply(extract_date)
        events = events.sort_values('event_date').reset_index(drop=True)
    except:
        # If sorting fails, continue without sorting
        pass

    # Display events
    for _, event in events.iterrows():
        options_list = event["options"].split(", ")

        # Different styling for car-free days
        is_car_free = "Car-Free Day" in event["title"]
        border_color = "#10b981" if is_car_free else "#2563eb"

        st.markdown(f"""
        <div style="padding: 10px; margin: 10px 0; border-radius: 8px;
background-color: #f9fafb; border-left: 5px solid {border_color};">
            <h4 style="margin-bottom: 5px;">{event["title"]}</h4>
            <p><strong>📍 Location:</strong> {event["location"]}</p>
            <p><strong>🕒 Time:</strong> {event["time"]}</p>
            <p><strong>🚦 Traffic Impact:</strong> {event["impact"]}</p>
            <p><strong>🔄 Alternative Options:</strong> {",
".join(options_list)}</p>
        </div>
        """, unsafe_allow_html=True)


    st.info("Note: Car-free days in Kigali are typically held on the 1st and
3rd Sunday of each month from 6am to 12pm. Always verify current information
with official city announcements.")
```

```
st.write("---")
st.markdown("<p style='text-align: center;'>Kigali Traffic Control System| © 2025</p>", unsafe_allow_html=True)
```