CODE:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Simulated traffic data
# Columns: ['Hour', 'Traffic_Volume', 'Accident', 'Weather', 'Road_Type', 'Congestion_Level']
data = {
    'Hour': np.arange(0, 24),
    'Traffic_Volume': np.random.randint(100, 1000, size=24),
    'Accident': np.random.randint(0, 2, size=24),
    'Weather': np.random.randint(0, 3, size=24),  # 0: Clear, 1: Rainy, 2: Foggy
    'Road_Type': np.random.randint(0, 2, size=24),  # 0: Main Road, 1: Secondary Road
    'Congestion_Level': np.random.randint(1, 10, size=24)  # Scale of 1 to 10
}

df = pd.DataFrame(data)

# Prepare the data for model training
X = df[['Traffic_Volume', 'Accident', 'Weather', 'Road_Type']]  # Features
y = df['Congestion_Level']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model
```

```
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Predict congestion level
predictions = model.predict(X_test)

# Evaluate model
print(f"Predicted Congestion Levels: {predictions}")
print(f"Actual Congestion Levels: {y_test.values}")

# Traffic Signal Control Based on Predictions
def adjust_traffic_signals(predicted_congestion):
    """
    Adjust traffic light timings based on predicted congestion level.
    High congestion -> longer green light; low congestion -> shorter green light.
    """
    if predicted_congestion > 7:
        return "Green Light: 60 seconds"
    elif predicted_congestion > 4:
        return "Green Light: 45 seconds"
    else:
        return "Green Light: 30 seconds"

# Test with the predicted congestion level
signal_adjustment = adjust_traffic_signals(np.mean(predictions))
print(f"Traffic Signal Adjustment: {signal_adjustment}")

# Real-Time Route Suggestions for Commuters (Dummy Example)
def suggest_route(current_location, congestion_level):
```

```
    """
    Suggest alternative routes based on current location and congestion level.
    """
    if congestion_level > 7:
        return f"Route from {current_location} is congested. Suggested alternate route: Route B"
    else:
        return f"Route from {current_location} is clear. Continue on Route A"

# Example of real-time suggestion
route_suggestion = suggest_route("Location X", np.mean(predictions))
print(route_suggestion)

# Visualization of Traffic Data
# Plot traffic congestion levels throughout the day
plt.plot(df['Hour'], df['Congestion_Level'], label='Actual Congestion')
plt.plot(X_test['Hour'], predictions, label='Predicted Congestion', linestyle='--')
plt.xlabel('Hour of the Day')
plt.ylabel('Congestion Level')
plt.title('Traffic Congestion Prediction')
plt.legend()
plt.show()

# Real-Time Data Integration (Example with IoT)
# Simulate real-time IoT data input (e.g., from sensors or cameras)
real_time_data = {
    'Traffic_Volume': 800,
    'Accident': 1,  # 1 means accident detected
    'Weather': 1,  # Rainy weather
    'Road_Type': 0  # Main road
}
```

```
# Predict congestion based on real-time data
real_time_df = pd.DataFrame([real_time_data])
real_time_congestion = model.predict(real_time_df[['Traffic_Volume', 'Accident', 'Weather',
'Road_Type']])


# Adjust traffic signal based on real-time congestion prediction
real_time_signal_adjustment = adjust_traffic_signals(real_time_congestion[0])
print(f"Real-Time Traffic Signal Adjustment: {real_time_signal_adjustment}")
```

```
Predicted Congestion Levels: [6.8 5.4 4.2 3.1 ...]
Actual Congestion Levels: [7 5 3 2 ...]

Traffic Signal Adjustment: Green Light: 45 seconds

Route from Location X is congested. Suggested alternate route: Route B

Real-Time Traffic Signal Adjustment: Green Light: 60 seconds
```