```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from traffic_ai_model import TrafficAIModel
from datetime import datetime, timedelta
import time

# Page configuration
st.set_page_config(
    page_title="Traffic AI Analysis",
    page_icon="🚗",
    layout="wide"
)

# Initialize session state
if "traffic_ai_model" not in st.session_state:
    st.session_state.traffic_ai_model = TrafficAIModel()

model = st.session_state.traffic_ai_model

# Main title
st.title("🚗 AI-Powered Traffic Flow Analysis")
st.markdown("---")

# Sidebar for navigation
st.sidebar.header("Navigation")
page = st.sidebar.selectbox(
    "Select Analysis Type",
    ["Traffic Prediction", "Congestion Analysis", "Route Optimization", "Traffic Insights"]
```

```
)

# Traffic Prediction Page
if page == "Traffic Prediction":
    st.header("🔮 Traffic Flow Prediction")

    # Generate sample data
    if st.button("Generate Traffic Data"):
        with st.spinner("Generating traffic data..."):
            data = model.create_traffic_data(5000)
            st.session_state.traffic_data = data
            st.success("Traffic data generated!")

    if "traffic_data" in st.session_state:
        data = st.session_state.traffic_data

        # Display data overview
        col1, col2 = st.columns([2, 1])

        with col1:
            st.subheader("Traffic Data Overview")
            st.dataframe(data.head())

            # Traffic flow over time
            fig = px.line(data.groupby('timestamp')['traffic_flow'].mean().reset_index(),
                          x='timestamp', y='traffic_flow',
                          title='Average Traffic Flow Over Time')
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("Data Statistics")
```

```
    st.write(data.describe())

    # Congestion distribution
    congestion_counts = data['congestion_level'].value_counts()
    fig = px.pie(values=congestion_counts.values, names=congestion_counts.index,
                 title='Congestion Level Distribution')
    st.plotly_chart(fig, use_container_width=True)

# Train traffic prediction model
if st.button("Train Traffic Prediction Model"):
    with st.spinner("Training traffic prediction model..."):
        results = model.train_traffic_prediction_model(data)
        st.session_state.traffic_results = results
        st.success("Traffic prediction model trained!")

if "traffic_results" in st.session_state:
    results = st.session_state.traffic_results

    # Display model performance
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("MSE", f"{results['mse']:,.2f}")
    with col2:
        st.metric("RMSE", f"{results['rmse']:,.2f}")
    with col3:
        st.metric("Model Type", "Random Forest")

    # Feature importance
    st.subheader("Feature Importance")
    importance_df = pd.DataFrame(
        list(results['feature_importance'].items()),
```

```
        columns=['Feature', 'Importance']
    ).sort_values('Importance', ascending=True)

    fig = px.bar(importance_df, x='Importance', y='Feature',
                 orientation='h', title='Feature Importance for Traffic Prediction')
    st.plotly_chart(fig, use_container_width=True)

    # Make predictions
    st.subheader("Predict Traffic Flow")

    col1, col2 = st.columns(2)
    with col1:
        hour = st.slider("Hour of Day", 0, 23, 12)
        day_of_week = st.selectbox("Day of Week",
                                   ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"])
        location = st.selectbox("Location",
                                ['Downtown', 'Highway A', 'Highway B', 'City Center', 'Suburbs',
                                 'Airport Road', 'University Area', 'Shopping District', 'Industrial Zone'])

    with col2:
        weather = st.selectbox("Weather Condition", ['Clear', 'Rain', 'Snow', 'Fog'])
        is_weekend = 1 if day_of_week in ["Saturday", "Sunday"] else 0
        is_rush_hour = 1 if (hour >= 7 and hour <= 9) or (hour >= 17 and hour <= 19) else 0

        day_mapping = {"Monday": 0, "Tuesday": 1, "Wednesday": 2, "Thursday": 3,
                       "Friday": 4, "Saturday": 5, "Sunday": 6}
        day_of_week_num = day_mapping[day_of_week]

    if st.button("Predict Traffic Flow"):
        input_data = {
```

```
                'hour': hour,
                'day_of_week': day_of_week_num,
                'is_weekend': is_weekend,
                'is_rush_hour': is_rush_hour,
                'location': location,
                'weather': weather
            }

            prediction = model.predict_traffic_flow(input_data)

            if isinstance(prediction, dict):
                st.success(f"Predicted Traffic Flow: {prediction['predicted_flow']:,.0f} vehicles/hour")
                st.info(f"Flow Category: {prediction['flow_category']}")
            else:
                st.error(prediction)

# Congestion Analysis Page
elif page == "Congestion Analysis":
    st.header("🚦 Congestion Analysis")

    if "traffic_data" in st.session_state:
        data = st.session_state.traffic_data

        # Train congestion model
        if st.button("Train Congestion Model"):
            with st.spinner("Training congestion prediction model..."):
                results = model.train_congestion_model(data)
                st.session_state.congestion_results = results
                st.success("Congestion model trained!")

        if "congestion_results" in st.session_state:
```

```
results = st.session_state.congestion_results

# Display model performance
col1, col2 = st.columns(2)
with col1:
    st.metric("Accuracy", f"{results['accuracy']:.2%}")
with col2:
    st.metric("Model Type", "Random Forest")

# Classification report
st.subheader("Classification Report")
st.text(results['classification_report'])

# Feature importance
st.subheader("Feature Importance for Congestion Prediction")
importance_df = pd.DataFrame(
    list(results['feature_importance'].items()),
    columns=['Feature', 'Importance']
).sort_values('Importance', ascending=True)

fig = px.bar(importance_df, x='Importance', y='Feature',
             orientation='h', title='Feature Importance for Congestion Prediction')
st.plotly_chart(fig, use_container_width=True)

# Congestion prediction interface
st.subheader("Predict Congestion Level")

col1, col2 = st.columns(2)
with col1:
    hour = st.slider("Hour of Day", 0, 23, 12, key="congestion_hour")
    location = st.selectbox("Location",
```

```
                    ['Downtown', 'Highway A', 'Highway B', 'City Center', 'Suburbs',
                     'Airport Road', 'University Area', 'Shopping District', 'Industrial Zone'],
                    key="congestion_location")
        weather = st.selectbox("Weather Condition", ['Clear', 'Rain', 'Snow', 'Fog'],
key="congestion_weather")

    with col2:
        traffic_flow = st.number_input("Current Traffic Flow", min_value=0, max_value=3000,
value=1000)
        day_of_week = st.selectbox("Day of Week",
                                   ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"],
                                   key="congestion_day")
        is_weekend = 1 if day_of_week in ["Saturday", "Sunday"] else 0
        is_rush_hour = 1 if (hour >= 7 and hour <= 9) or (hour >= 17 and hour <= 19) else 0

        day_mapping = {"Monday": 0, "Tuesday": 1, "Wednesday": 2, "Thursday": 3,
                "Friday": 4, "Saturday": 5, "Sunday": 6}
        day_of_week_num = day_mapping[day_of_week]

    if st.button("Predict Congestion"):
        input_data = {
            'hour': hour,
            'day_of_week': day_of_week_num,
            'is_weekend': is_weekend,
            'is_rush_hour': is_rush_hour,
            'traffic_flow': traffic_flow,
            'location': location,
            'weather': weather
        }

        prediction = model.predict_congestion(input_data)
```

```
            if isinstance(prediction, dict):
                if prediction['predicted_congestion'] == 'High':
                    st.error(f"Predicted Congestion: {prediction['predicted_congestion']} (Confidence: {prediction['confidence']:.2%})")
                elif prediction['predicted_congestion'] == 'Medium':
                    st.warning(f"Predicted Congestion: {prediction['predicted_congestion']} (Confidence: {prediction['confidence']:.2%})")
                else:
                    st.success(f"Predicted Congestion: {prediction['predicted_congestion']} (Confidence: {prediction['confidence']:.2%})")

                # Show all probabilities
                st.subheader("Congestion Probabilities")
                for level, prob in prediction['all_probabilities'].items():
                    st.write(f"{level}: {prob:.2%}")
            else:
                st.error(prediction)

# Route Optimization Page
elif page == "Route Optimization":
    st.header("🗺️ Route Optimization")

    st.subheader("Find Optimal Route")

    col1, col2 = st.columns(2)
    with col1:
        start_location = st.selectbox("Start Location",
                                      ['Downtown', 'Highway A', 'Highway B', 'City Center', 'Suburbs',
                                       'Airport Road', 'University Area', 'Shopping District', 'Industrial Zone'])
        current_time = st.time_input("Current Time", value=datetime.now().time())
```

```
with col2:
    end_location = st.selectbox("End Location",
                        ['Downtown', 'Highway A', 'Highway B', 'City Center', 'Suburbs',
                         'Airport Road', 'University Area', 'Shopping District', 'Industrial Zone'])
    weather = st.selectbox("Weather Condition", ['Clear', 'Rain', 'Snow', 'Fog'])

if st.button("Optimize Route"):
    with st.spinner("Calculating optimal routes..."):
        routes = model.optimize_route(start_location, end_location, current_time, weather)

        st.subheader("Recommended Routes")

        for i, route in enumerate(routes):
            with st.container():
                col1, col2, col3, col4 = st.columns(4)

                with col1:
                    st.metric("Route", route['name'])
                with col2:
                    st.metric("Distance", f"{route['distance']} km")
                with col3:
                    st.metric("Time", f"{route['estimated_time']} min")
                with col4:
                    if route['congestion'] == 'High':
                        st.error(f"Congestion: {route['congestion']}")
                    elif route['congestion'] == 'Medium':
                        st.warning(f"Congestion: {route['congestion']}")
                    else:
                        st.success(f"Congestion: {route['congestion']}")

                st.divider()
```

```
# Traffic Insights Page
elif page == "Traffic Insights":
    st.header("📊 Traffic Insights")

    if "traffic_data" in st.session_state:
        data = st.session_state.traffic_data

        # Generate insights
        insights = model.get_traffic_insights(data)

        # Display insights
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Peak Hours")
            peak_hours = insights['peak_hours']
            st.write(f"Busiest hours: {peak_hours[0]}:00, {peak_hours[1]}:00, {peak_hours[2]}:00")

            # Peak hours visualization
            hourly_flow = data.groupby('hour')['traffic_flow'].mean()
            fig = px.bar(x=hourly_flow.index, y=hourly_flow.values,
                         title='Average Traffic Flow by Hour',
                         labels={'x': 'Hour of Day', 'y': 'Average Traffic Flow'})
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("Busiest Locations")
            busiest_locations = insights['busiest_locations']
            st.write(f"Busiest areas: {', '.join(busiest_locations)}")
```

```
        # Location traffic visualization
        location_flow = data.groupby('location')['traffic_flow'].mean()
        fig = px.bar(x=location_flow.index, y=location_flow.values,
                     title='Average Traffic Flow by Location',
                     labels={'x': 'Location', 'y': 'Average Traffic Flow'})
        st.plotly_chart(fig, use_container_width=True)

    # Congestion by day
    st.subheader("Congestion by Day of Week")
    congestion_by_day = insights['congestion_by_day']
    day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    congestion_values = [congestion_by_day.get(i, 0) for i in range(7)]

    fig = px.bar(x=day_names, y=congestion_values,
                 title='Percentage of High Congestion by Day',
                 labels={'x': 'Day of Week', 'y': 'High Congestion %'})
    st.plotly_chart(fig, use_container_width=True)

    # Weather impact
    st.subheader("Weather Impact on Traffic")
    weather_impact = insights['weather_impact']
    weather_df = pd.DataFrame(list(weather_impact.items()), columns=['Weather', 'Average Flow'])

    fig = px.bar(weather_df, x='Weather', y='Average Flow',
                 title='Average Traffic Flow by Weather Condition')
    st.plotly_chart(fig, use_container_width=True)

# Model management
st.sidebar.markdown("---")
st.sidebar.header("Model Management")
```

```
if st.sidebar.button("Save Models"):
    if model.traffic_model is not None:
        result = model.save_models()
        st.sidebar.success(result)
    else:
        st.sidebar.error("No trained models to save")

if st.sidebar.button("Load Models"):
    result = model.load_models("traffic_models.pkl")
    st.sidebar.success(result)

# Information section
with st.expander("ℹ️ About Traffic AI"):
    st.write("""
    **AI-Powered Traffic Analysis System:**

    This system uses machine learning to analyze and predict traffic patterns:

    **Traffic Prediction:**
    - Predicts traffic flow based on time, location, and weather
    - Uses Random Forest regression for accurate predictions
    - Considers rush hours, weekends, and weather conditions

    **Congestion Analysis:**
    - Classifies congestion levels (Low, Medium, High)
    - Provides confidence scores for predictions
    - Helps identify problematic areas and times

    **Route Optimization:**
    - Suggests optimal routes based on current conditions
    - Considers traffic levels, weather, and time of day
```

- Provides multiple route options with estimated times

Traffic Insights:

- Identifies peak hours and busy locations
- Analyzes weather impact on traffic
- Shows congestion patterns by day of week

Key Features:

- Real-time predictions
- Interactive visualizations
- Model persistence and loading
- Comprehensive traffic analysis

""")

 # Footer

st.markdown("---")

st.markdown("🚗 Built with Streamlit and Scikit-learn | AI-Powered Traffic Analysis")