In [1]:

```
!pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in /Users/joeynewfield/anaconda3/lib/python3.11/site-pa
ckages (8.0.4)
Requirement already satisfied: ipykernel>=4.5.1 in /Users/joeynewfield/anaconda3/lib/python3.11/s
ite-packages (from ipywidgets) (6.19.2)
Requirement already satisfied: ipython>=6.1.0 in /Users/joeynewfield/anaconda3/lib/python3.11/sit
e-packages (from ipywidgets) (8.12.0)
Requirement already satisfied: traitlets>=4.3.1 in /Users/joeynewfield/anaconda3/lib/python3.11/s
ite-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=4.0 in /Users/joeynewfield/anaconda3/lib/pytho
n3.11/site-packages (from ipywidgets) (4.0.5)
Requirement already satisfied: jupyterlab-widgets~=3.0 in /Users/joeynewfield/anaconda3/lib/pytho
n3.11/site-packages (from ipywidgets) (3.0.5)
Requirement already satisfied: appnope in /Users/joeynewfield/anaconda3/lib/python3.11/site-packa
ges (from ipykernel>=4.5.1->ipywidgets) (0.1.2)
Requirement already satisfied: comm>=0.1.1 in /Users/joeynewfield/anaconda3/lib/python3.11/site-p
ackages (from ipykernel>=4.5.1->ipywidgets) (0.1.2)
Requirement already satisfied: debugpy>=1.0 in /Users/joeynewfield/anaconda3/lib/python3.11/site-
packages (from ipykernel>=4.5.1->ipywidgets) (1.6.7)
Requirement already satisfied: jupyter-client>=6.1.12 in /Users/joeynewfield/anaconda3/lib/python
3.11/site-packages (from ipykernel>=4.5.1->ipywidgets) (7.4.9)
Requirement already satisfied: matplotlib-inline>=0.1 in /Users/joeynewfield/anaconda3/lib/python
3.11/site-packages (from ipykernel>=4.5.1->ipywidgets) (0.1.6)
Requirement already satisfied: nest-asyncio in /Users/joeynewfield/anaconda3/lib/python3.11/site-
packages (from ipykernel>=4.5.1->ipywidgets) (1.5.6)
Requirement already satisfied: packaging in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from ipykernel>=4.5.1->ipywidgets) (23.0)
Requirement already satisfied: psutil in /Users/joeynewfield/anaconda3/lib/python3.11/site-packag
es (from ipykernel>=4.5.1->ipywidgets) (5.9.0)
Requirement already satisfied: pyzmq>=17 in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from ipykernel>=4.5.1->ipywidgets) (23.2.0)
Requirement already satisfied: tornado>=6.1 in /Users/joeynewfield/anaconda3/lib/python3.11/site-
packages (from ipykernel>=4.5.1->ipywidgets) (6.3.2)
Requirement already satisfied: backcall in /Users/joeynewfield/anaconda3/lib/python3.11/site-pack
ages (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: decorator in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: jedi>=0.16 in /Users/joeynewfield/anaconda3/lib/python3.11/site-pa
ckages (from ipython>=6.1.0->ipywidgets) (0.18.1)
Requirement already satisfied: pickleshare in /Users/joeynewfield/anaconda3/lib/python3.11/site-p
ackages (from ipython>=6.1.0->ipywidgets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in /Users/joeynewfield/anac
```

```
onda3/lib/python3.11/site-packages (from ipython>=6.1.0->ipywidgets) (3.0.36)
Requirement already satisfied: pygments>=2.4.0 in /Users/joeynewfield/anaconda3/lib/python3.11/si
te-packages (from ipython>=6.1.0->ipywidgets) (2.15.1)
Requirement already satisfied: stack-data in /Users/joeynewfield/anaconda3/lib/python3.11/site-pa
ckages (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: pexpect>4.3 in /Users/joeynewfield/anaconda3/lib/python3.11/site-p
ackages (from ipython>=6.1.0->ipywidgets) (4.8.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /Users/joeynewfield/anaconda3/lib/python3.1
1/site-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: entrypoints in /Users/joeynewfield/anaconda3/lib/python3.11/site-p
ackages (from jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (0.4)
Requirement already satisfied: jupyter-core>=4.9.2 in /Users/joeynewfield/anaconda3/lib/python3.1
1/site-packages (from jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (5.3.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/joeynewfield/anaconda3/lib/python
3.11/site-packages (from jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (2.8.2)
Requirement already satisfied: ptyprocess>=0.5 in /Users/joeynewfield/anaconda3/lib/python3.11/si
te-packages (from pexpect>4.3->ipython>=6.1.0->ipywidgets) (0.7.0)
Requirement already satisfied: wcwidth in /Users/joeynewfield/anaconda3/lib/python3.11/site-packa
ges (from prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30->ipython>=6.1.0->ipywidgets) (0.2.5)
Requirement already satisfied: executing in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from stack-data->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: asttokens in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from stack-data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in /Users/joeynewfield/anaconda3/lib/python3.11/site-pac
kages (from stack-data->ipython>=6.1.0->ipywidgets) (0.2.2)
Requirement already satisfied: platformdirs>=2.5 in /Users/joeynewfield/anaconda3/lib/python3.11/
site-packages (from jupyter-core>=4.9.2->jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets)
(2.5.2)
Requirement already satisfied: six>=1.5 in /Users/joeynewfield/anaconda3/lib/python3.11/site-pack
ages (from python-dateutil>=2.8.2->jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (1.16.0)
```

In [8]:

```
import pandas as pd
import numpy as np
import joblib
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output

class TrafficDashboard:
    def __init__(self):
        # Load models and scalers
        print("Loading models and scalers...")
        self.dt_model = joblib.load('decision_tree_model.joblib')
        self.lr_model = joblib.load('linear_regression_model.joblib')
        self.dt_scaler = joblib.load('dt_scaler.joblib')
        self.lr_scaler = joblib.load('lr_scaler.joblib')

        # Load historical data for better feature calculation
        self.historical_data = pd.read_csv('engineered_traffic_data.csv')

        # Create widgets
        self.create_widgets()

    def create_widgets(self):
        # Time inputs
        self.hour_widget = widgets.IntSlider(
            value=12,
            min=0,
            max=23,
            description='Hour:',
            style={'description_width': 'initial'}
        )

        self.is_weekend_widget = widgets.Checkbox(
            value=False,
            description='Is Weekend',
            style={'description_width': 'initial'}
        )

        # Weather inputs
        self.temperature_widget = widgets.FloatSlider(
            value=70.0,
            min=30.0,
            max=100.0,
            description='Temperature (°F):',
            style={'description_width': 'initial'}
        )

        self.humidity_widget = widgets.IntSlider(
            value=50,
            min=0,
            max=100,
            description='Humidity (%):',
            style={'description_width': 'initial'}
        )

        self.wind_speed_widget = widgets.FloatSlider(
            value=5.0,
            min=0.0,
            max=30.0,
            description='Wind Speed (mph):',
            style={'description_width': 'initial'}
        )

        self.visibility_widget = widgets.IntSlider(
            value=5000,
            min=0,
            max=10000,
            description='Visibility (m):',
            style={'description_width': 'initial'}
        )

        self.precipitation_widget = widgets.FloatSlider(
            value=0.0,
            min=0.0,
            max=50.0,
            description='Precipitation (mm):',
            style={'description_width': 'initial'}
        )

        self.weather_widget = widgets.Dropdown(
            options=['Clear', 'Clouds', 'Mist', 'Fog', 'Rain', 'Snow', 'Thunderstorm'],
            value='Clear',
            description='Weather:',
            style={'description_width': 'initial'}
        )

        # Traffic history inputs (with more reasonable ranges)
        avg_traffic = self.historical_data['traffic_count'].mean()
        self.rolling_avg_widget = widgets.IntSlider(
            value=int(avg_traffic),
            min=0,
            max=2000,
            description='3-Hr Rolling Avg:',
            style={'description_width': 'initial'}
        )

        self.traffic_density_widget = widgets.FloatSlider(
            value=0.05,  # Typical value from historical data
            min=0.0,
            max=0.2,
            step=0.01,
            description='Traffic Density:',
            style={'description_width': 'initial'}
        )

        # Update button
        self.update_button = widgets.Button(
            description='Update Predictions',
            style={'description_width': 'initial'}
        )
        self.update_button.on_click(self.update_predictions)

        # Output widget for plots
        self.output = widgets.Output()

    def get_typical_values(self, hour, is_weekend):
        """Get typical traffic values for given hour and day type."""
        mask = (self.historical_data['hour'] == hour) & \
               (self.historical_data['is_weekend'] == is_weekend)

        typical_data = self.historical_data[mask]

        return {
            'rolling_avg': typical_data['rolling_avg_3h'].median(),
            'traffic_density': typical_data['traffic_density'].median(),
        }

    def create_feature_vector(self):
        """Create feature vector from widget values with better scaling."""
        hour = self.hour_widget.value
        is_weekend = self.is_weekend_widget.value

        # Get typical values for current conditions
        typical_values = self.get_typical_values(hour, is_weekend)

        # Use these for scaling if widget values aren't set
        rolling_avg = self.rolling_avg_widget.value or typical_values['rolling_avg']
        traffic_density = self.traffic_density_widget.value or typical_values['traffic_density']

        weather_severity = {
            'Clear': 0, 'Clouds': 1, 'Mist': 2, 'Fog': 3,
            'Rain': 4, 'Snow': 5, 'Thunderstorm': 6
        }

        features = {
            'hour_sin': np.sin(2 * np.pi * hour/24),
            'hour_cos': np.cos(2 * np.pi * hour/24),
            'is_morning_peak': 1 if 6 <= hour <= 9 else 0,
            'is_evening_peak': 1 if 16 <= hour <= 19 else 0,
            'is_weekend': int(is_weekend),
            'day_of_week_num': 6 if is_weekend else 2,  # Sample weekday/weekend
            'month': datetime.now().month,
            'temperature': self.temperature_widget.value,
            'humidity': self.humidity_widget.value,
            'wind_speed': self.wind_speed_widget.value,
            'visibility': self.visibility_widget.value,
            'precipitation': self.precipitation_widget.value,
            'weather_severity': weather_severity[self.weather_widget.value],
            'rolling_avg_3h': rolling_avg,
            'traffic_density': traffic_density
        }

        return pd.DataFrame([features])

    def update_predictions(self, _):
        """Update predictions and plots."""
        with self.output:
            clear_output(wait=True)

            # Get feature vector
            features = self.create_feature_vector()

            # Make predictions
            X_scaled_dt = self.dt_scaler.transform(features)
            X_scaled_lr = self.lr_scaler.transform(features)

            congestion_level = self.dt_model.predict(X_scaled_dt)[0]
            traffic_count = self.lr_model.predict(X_scaled_lr)[0]

            # Ensure predictions are within reasonable bounds
            traffic_count = np.clip(traffic_count, 0, 2000)

            # Create figure with two subplots
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

            # Plot 1: Current predictions
            ax1.bar(['Traffic Volume'], [traffic_count], color='skyblue')
            ax1.set_title('Predicted Traffic Volume')
            ax1.set_ylabel('Vehicles per Hour')

            # Plot 2: 24-hour prediction with historical context
            hours = range(24)
            predictions = []
            historical_avg = []

            for h in hours:
                # Get predictions
                features.loc[0, 'hour_sin'] = np.sin(2 * np.pi * h/24)
                features.loc[0, 'hour_cos'] = np.cos(2 * np.pi * h/24)
                features.loc[0, 'is_morning_peak'] = 1 if 6 <= h <= 9 else 0
                features.loc[0, 'is_evening_peak'] = 1 if 16 <= h <= 19 else 0

                # Update features with typical values for that hour
                typical_values = self.get_typical_values(h, self.is_weekend_widget.value)
                features.loc[0, 'rolling_avg_3h'] = typical_values['rolling_avg']
                features.loc[0, 'traffic_density'] = typical_values['traffic_density']

                X_scaled = self.lr_scaler.transform(features)
                pred = self.lr_model.predict(X_scaled)[0]
                predictions.append(np.clip(pred, 0, 2000))

                # Get historical average for context
                mask = (self.historical_data['hour'] == h) & \
                       (self.historical_data['is_weekend'] == self.is_weekend_widget.value)
                hist_avg = self.historical_data[mask]['traffic_count'].mean()
                historical_avg.append(hist_avg)

            ax2.plot(hours, predictions, marker='o', label='Predicted')
            ax2.plot(hours, historical_avg, '--', label='Historical Average', alpha=0.5)
            ax2.set_title('24-Hour Traffic Prediction')
            ax2.set_xlabel('Hour of Day')
            ax2.set_ylabel('Predicted Traffic Volume')
            ax2.legend()

            plt.tight_layout()
            plt.show()

            # Display numeric predictions
            congestion_labels = ['Light', 'Moderate', 'High', 'Severe']
            print(f"\nPredictions for Hour {self.hour_widget.value}:00")
            print(f"Traffic Volume: {int(traffic_count)} vehicles/hour")
            print(f"Congestion Level: {congestion_labels[int(congestion_level)]}")

            # Show historical context
            mask = (self.historical_data['hour'] == self.hour_widget.value) & \
                   (self.historical_data['is_weekend'] == self.is_weekend_widget.value)
            hist_avg = self.historical_data[mask]['traffic_count'].mean()
            print(f"\nHistorical average for this hour: {int(hist_avg)} vehicles/hour")

    def display_dashboard(self):
        """Display the dashboard."""
        # Create layout
        input_widgets = widgets.VBox([
            widgets.HBox([self.hour_widget, self.is_weekend_widget]),
            widgets.HBox([self.temperature_widget, self.humidity_widget]),
            widgets.HBox([self.wind_speed_widget, self.visibility_widget]),
            widgets.HBox([self.precipitation_widget, self.weather_widget]),
            widgets.HBox([self.rolling_avg_widget, self.traffic_density_widget]),
            self.update_button
        ])
        # Display dashboard
        display(input_widgets)
        display(self.output)

        # Initial update
        self.update_predictions(None)

# Create and display dashboard
print("Initializing Traffic Prediction Dashboard...")
dashboard = TrafficDashboard()
dashboard.display_dashboard()
```

```
Initializing Traffic Prediction Dashboard...
Loading models and scalers...
```

Hour: ○ 12 □ Is Weekend

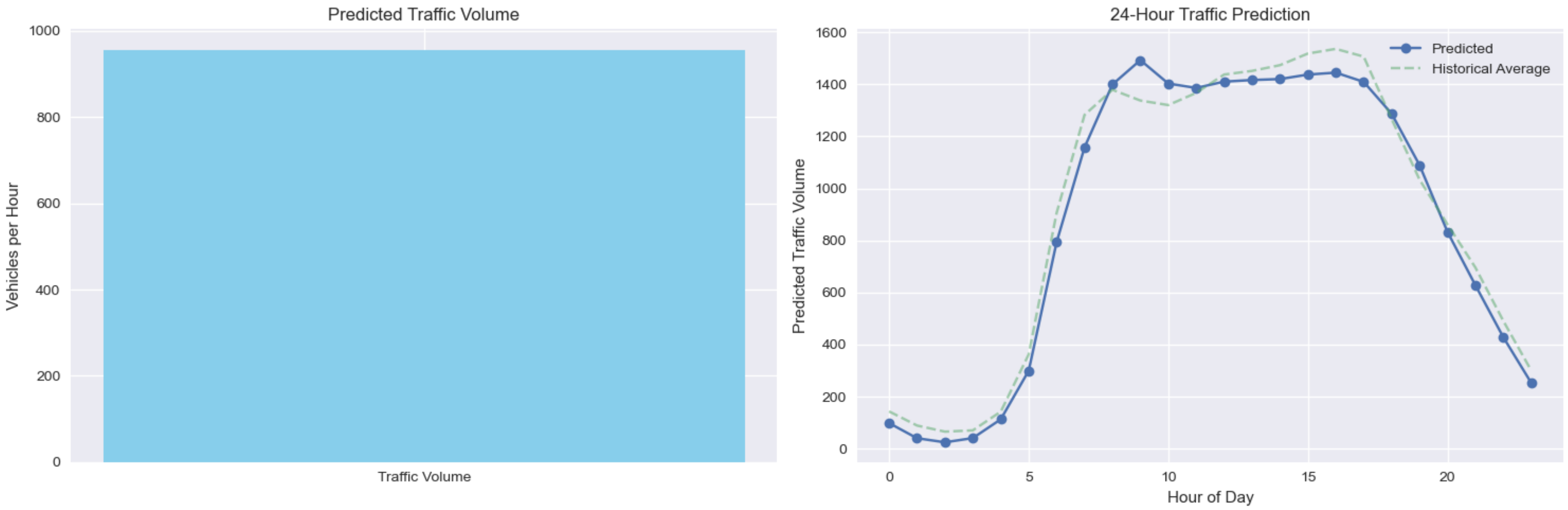
Temperature (°F): ○ 70.00 Humidity (%): ○ 50

Wind Speed (mph): ○ 5.00 Visibility (m): ○ 5000

Precipitation (mm): ○ 0.00 Weather: Clear

3-Hr Rolling Avg: ○ 900 Traffic Density: ○ 0.05

Update Predictions



```
Predictions for Hour 12:00
Traffic Volume: 956 vehicles/hour
Congestion Level: High

Historical average for this hour: 1437 vehicles/hour
```

In [ ]: