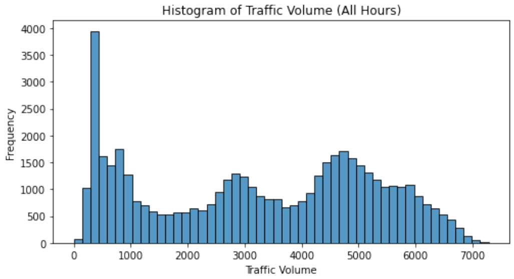
In [1]:

```python
# ==============================================================================
# Indicators of Heavy Traffic on I-94: Enhanced Analysis with Interactive Dashboard
# ==============================================================================

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

# For the interactive dashboard in a Jupyter Notebook
from jupyter_dash import JupyterDash
from dash import dcc, html, Input, Output
import plotly.express as px

# ------------------------------------------------------------------------------
# 1. Data Loading and Initial Exploration
# ------------------------------------------------------------------------------

# Load the dataset (download from UCI Machine Learning Repository if needed)
i_94 = pd.read_csv('Metro_Interstate_Traffic_Volume.csv')
print("Initial Data Snapshot:")
print(i_94.head())
print("\nDataset Information:")
print(i_94.info())

# ------------------------------------------------------------------------------
# 2. Data Cleaning and Feature Engineering
# ------------------------------------------------------------------------------

# -- 2.1 Holiday Column --
# Fill missing holiday values with a default 'No Holiday'
i_94['holiday'] = i_94['holiday'].fillna('No Holiday')

# Create a binary indicator for holidays (1 if holiday, else 0)
i_94['is_holiday'] = i_94['holiday'].apply(lambda x: 0 if x == 'No Holiday' else 1)

# -- 2.2 Date-Time Conversion and Feature Extraction --
# Convert date_time column to datetime type
i_94['date_time'] = pd.to_datetime(i_94['date_time'])

# Extract additional time-based features
i_94['year'] = i_94['date_time'].dt.year
i_94['month'] = i_94['date_time'].dt.month
i_94['day'] = i_94['date_time'].dt.day
i_94['hour'] = i_94['date_time'].dt.hour
i_94['dayofweek'] = i_94['date_time'].dt.dayofweek  # Monday=0, Sunday=6
i_94['is_weekend'] = i_94['dayofweek'].apply(lambda x: 1 if x >= 5 else 0)

# Define rush hours: morning (7-9 AM) and evening (4-6 PM)
def is_rush(hour):
    return 1 if (7 <= hour < 10) or (16 <= hour < 19) else 0

i_94['is_rush_hour'] = i_94['hour'].apply(is_rush)

# -- 2.3 Check for Missing Hours --
# Set date_time as index to check for missing hours over the entire period
i_94_indexed = i_94.set_index('date_time').sort_index()
full_range = pd.date_range(start=i_94_indexed.index.min(), end=i_94_indexed.index.max(), freq='H')
missing_hours = full_range.difference(i_94_indexed.index)
print(f"\nMissing hours in the dataset: {len(missing_hours)}")
# (Optional) Create a DataFrame of missing hours to inspect them:
missing_df = pd.DataFrame(missing_hours, columns=['missing_date_time'])
print(missing_df.head())

# ------------------------------------------------------------------------------
# 3. Exploratory Data Analysis (EDA)
# ------------------------------------------------------------------------------

# -- 3.1 Overall Traffic Volume Distribution --
plt.figure(figsize=(8, 4))
sns.histplot(i_94['traffic_volume'], bins=50, kde=False)
plt.title('Histogram of Traffic Volume (All Hours)')
plt.xlabel('Traffic Volume')
plt.ylabel('Frequency')
plt.show()

print("\nTraffic Volume Summary Statistics:")
print(i_94['traffic_volume'].describe())

# -- 3.2 Day vs. Night Analysis --
# Define daytime: 7 AM to 7 PM; nighttime: 7 PM to 7 AM
day = i_94[(i_94['hour'] >= 7) & (i_94['hour'] < 19)]
night = i_94[(i_94['hour'] >= 19) | (i_94['hour'] < 7)]

print(f"\nDaytime data shape: {day.shape}")
print(f"Nighttime data shape: {night.shape}")

# Plot histograms for day and night traffic volume
plt.figure(figsize=(11, 4))

plt.subplot(1, 2, 1)
plt.hist(day['traffic_volume'], bins=50)
plt.xlim(-100, 7500)
plt.ylim(0, 8000)
plt.title('Traffic Volume: Day')
plt.xlabel('Traffic Volume')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(night['traffic_volume'], bins=50)
plt.xlim(-100, 7500)
plt.ylim(0, 8000)
plt.title('Traffic Volume: Night')
plt.xlabel('Traffic Volume')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# -- 3.3 Time Indicators --
# a) Traffic Volume by Month (using daytime data)
by_month = day.groupby('month')['traffic_volume'].mean()
plt.figure(figsize=(8, 4))
by_month.plot.line(marker='o')
plt.title('Average Daytime Traffic Volume by Month')
plt.xlabel('Month')
plt.ylabel('Average Traffic Volume')
plt.grid(True)
plt.show()

# b) Traffic Volume by Day of Week
by_dayofweek = day.groupby('dayofweek')['traffic_volume'].mean()
plt.figure(figsize=(8, 4))
by_dayofweek.plot.line(marker='o')
```

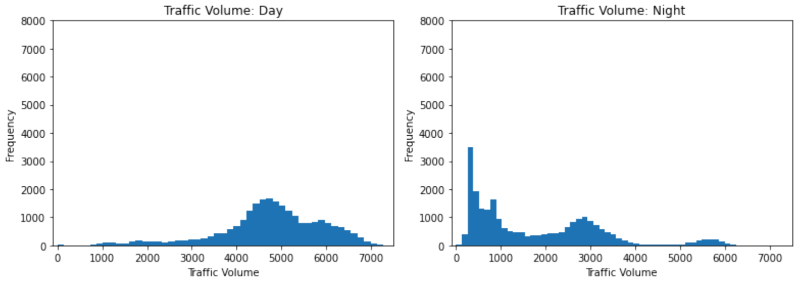
```python
plt.title('Average Daytime Traffic Volume by Day of Week')
plt.xlabel('Day of Week (0=Monday)')
plt.ylabel('Average Traffic Volume')
plt.grid(True)
plt.show()

# c) Traffic Volume by Hour on Business Days vs. Weekends
business_days = day[day['is_weekend'] == 0]
weekend = day[day['is_weekend'] == 1]

# Group by hour for each subset
by_hour_business = business_days.groupby('hour')['traffic_volume'].mean()
by_hour_weekend = weekend.groupby('hour')['traffic_volume'].mean()

plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
by_hour_business.plot.line(marker='o')
plt.xlim(6, 20)
plt.ylim(1500, 6500)
plt.title('Traffic Volume by Hour: Business Days')
plt.xlabel('Hour')
plt.ylabel('Traffic Volume')
plt.grid(True)

plt.subplot(1, 2, 2)
by_hour_weekend.plot.line(marker='o')
plt.xlim(6, 20)
plt.ylim(1500, 6500)
plt.title('Traffic Volume by Hour: Weekends')
plt.xlabel('Hour')
plt.ylabel('Traffic Volume')
plt.grid(True)
plt.tight_layout()
plt.show()

# -----------------------------------------------------------------------------
# 4. Time Series Decomposition and Correlation Analysis
# -----------------------------------------------------------------------------

# -- 4.1 Time Series Decomposition --
# Aggregate traffic volume by day (summing the hourly volumes) for decomposition
daily_traffic = i_94_indexed['traffic_volume'].resample('D').sum()
decomposition = seasonal_decompose(daily_traffic, model='additive', period=365)
fig = decomposition.plot()
fig.set_size_inches(14, 10)
plt.suptitle('Seasonal Decomposition of Daily Traffic Volume', fontsize=16)
plt.show()

# -- 4.2 Correlation Heatmap of Numeric Features --
numeric_cols = i_94.select_dtypes(include=['float64', 'int64'])
corr_matrix = numeric_cols.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numeric Features')
plt.show()

# -- 4.3 Outlier Detection (Using Z-Score) --
# Calculate z-scores for traffic_volume (daytime data)
day['traffic_zscore'] = np.abs(stats.zscore(day['traffic_volume']))
outliers = day[day['traffic_zscore'] > 3]
print(f"\nNumber of outlier hours (z-score > 3) in daytime data: {outliers.shape[0]}")

# Plot outlier traffic volumes
plt.figure(figsize=(10, 4))
plt.plot(day['date_time'], day['traffic_volume'], label='Traffic Volume')
plt.scatter(outliers['date_time'], outliers['traffic_volume'], color='red', label='Outliers')
plt.xlabel('Date')
plt.ylabel('Traffic Volume')
plt.title('Daytime Traffic Volume with Outliers Highlighted')
plt.legend()
plt.show()

# -----------------------------------------------------------------------------
# 5. Weather Analysis
# -----------------------------------------------------------------------------

# -- 5.1 Scatter Plot: Temperature vs. Traffic Volume --
plt.figure(figsize=(8, 4))
sns.scatterplot(x='traffic_volume', y='temp', data=day, alpha=0.5)
plt.ylim(230, 320)  # limit y-axis to avoid outliers from erroneous temperatures
plt.title('Scatter Plot: Temperature vs. Traffic Volume (Daytime)')
plt.xlabel('Traffic Volume')
plt.ylabel('Temperature (K)')
plt.show()

# -- 5.2 Average Traffic Volume by Weather Categories --
# a) Group by weather_main
by_weather_main = day.groupby('weather_main')['traffic_volume'].mean()
plt.figure(figsize=(8, 4))
by_weather_main.sort_values().plot.barh(color='skyblue')
plt.title('Average Traffic Volume by Weather Main')
plt.xlabel('Average Traffic Volume')
plt.show()

# b) Group by weather_description (more granular)
by_weather_description = day.groupby('weather_description')['traffic_volume'].mean()
plt.figure(figsize=(6, 10))
by_weather_description.sort_values().plot.barh(color='salmon')
plt.title('Average Traffic Volume by Weather Description')
plt.xlabel('Average Traffic Volume')
plt.show()

# -----------------------------------------------------------------------------
# 6. Modeling: Predicting Heavy Traffic Periods
# -----------------------------------------------------------------------------

# -- 6.1 Define Heavy Traffic --
# For this analysis, we define heavy traffic as traffic volume above the 75th percentile in daytime data.
heavy_threshold = day['traffic_volume'].quantile(0.75)
print(f"\nHeavy traffic threshold (75th percentile of daytime): {heavy_threshold:.0f} cars/hour")
day['heavy_traffic'] = day['traffic_volume'].apply(lambda x: 1 if x > heavy_threshold else 0)

# -- 6.2 Prepare Features for Modeling --
# Select features: time-based and weather-based features
features = ['hour', 'dayofweek', 'month', 'temp', 'rain_1h', 'snow_1h', 'clouds_all', 'is_holiday', 'is_rush_hour']
X = day[features]
y = day['heavy_traffic']

# -- 6.3 Split Data and Train a Model --
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Use a Random Forest Classifier for prediction
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# -- 6.4 Evaluate the Model --
y_pred = rf.predict(X_test)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# -- 6.5 Feature Importance --
feat_importances = pd.Series(rf.feature_importances_, index=features).sort_values(ascending=False)
plt.figure(figsize=(8, 4))
```

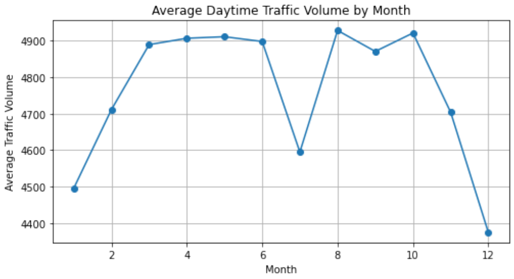
```python
feat_importances.plot.bar(color='mediumseagreen')
plt.title('Feature Importances for Heavy Traffic Prediction')
plt.ylabel('Importance')
plt.show()

# -----------------------------------------------------------------------------
# 7. Enhanced Visualization and Interactive Dashboard
# -----------------------------------------------------------------------------

# For the dashboard, we need a DataFrame variable named 'df'.
# We'll use the processed DataFrame from earlier.
df = i_94.copy()

# Prepare dropdown options for the dashboard filters
weather_options = [{'label': w, 'value': w} for w in sorted(df['weather_main'].unique())]
day_options = [{'label': f"{d} ({['Mon','Tue','Wed','Thu','Fri','Sat','Sun'][d]})", 'value': d}
               for d in sorted(df['dayofweek'].unique())]

# Create the JupyterDash app (for running inside a Jupyter Notebook)
app = JupyterDash(__name__)

app.layout = html.Div([
    html.H1("I-94 Traffic Analysis Dashboard", style={'textAlign': 'center'}),

    # Filter Controls
    html.Div([
        html.Div([
            html.Label("Select Weather Condition (weather_main):"),
            dcc.Dropdown(
                id='weather-dropdown',
                options=weather_options,
                multi=True,
                placeholder="Select weather conditions"
            )
        ], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'}),

        html.Div([
            html.Label("Select Day of Week:"),
            dcc.Dropdown(
                id='dayofweek-dropdown',
                options=day_options,
                multi=True,
                placeholder="Select day(s) (0=Mon, 6=Sun)"
            )
        ], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'}),

        html.Div([
            html.Label("Rush Hour Filter:"),
            dcc.RadioItems(
                id='rush-hour-radio',
                options=[
                    {'label': 'All Hours', 'value': 'all'},
                    {'label': 'Rush Hour Only', 'value': 'rush'},
                    {'label': 'Non-Rush Hour Only', 'value': 'non_rush'}
                ],
                value='all',
                labelStyle={'display': 'inline-block', 'margin-right': '10px'}
            )
        ], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'}),
    ], style={'display': 'flex', 'justifyContent': 'center'}),

    # Date Picker for custom date range filtering
    html.Div([
        html.Label("Select Date Range:"),
        dcc.DatePickerRange(
            id='date-picker-range',
            min_date_allowed=df['date_time'].min().date(),
            max_date_allowed=df['date_time'].max().date(),
            start_date=df['date_time'].min().date(),
            end_date=df['date_time'].max().date()
        )
    ], style={'padding': '10px', 'textAlign': 'center'}),

    # Graph Output
    dcc.Graph(id='time-series-graph')
])

# Callback to update the time series plot based on the selected filters
@app.callback(
    Output('time-series-graph', 'figure'),
    [Input('weather-dropdown', 'value'),
     Input('dayofweek-dropdown', 'value'),
     Input('rush-hour-radio', 'value'),
     Input('date-picker-range', 'start_date'),
     Input('date-picker-range', 'end_date')]
)
def update_graph(selected_weather, selected_days, rush_filter, start_date, end_date):
    # Start with the full dataframe
    filtered_df = df.copy()

    # Filter by date range
    filtered_df = filtered_df[(filtered_df['date_time'] >= start_date) & (filtered_df['date_time'] <= end_date)]

    # Filter by weather condition if selections exist
    if selected_weather:
        filtered_df = filtered_df[filtered_df['weather_main'].isin(selected_weather)]

    # Filter by day of week if selections exist
    if selected_days:
        filtered_df = filtered_df[filtered_df['dayofweek'].isin(selected_days)]

    # Apply rush hour filter
    if rush_filter == 'rush':
        filtered_df = filtered_df[filtered_df['is_rush_hour'] == 1]
    elif rush_filter == 'non_rush':
        filtered_df = filtered_df[filtered_df['is_rush_hour'] == 0]

    # Create a line plot of traffic volume over time using Plotly Express
    fig = px.line(
        filtered_df.sort_values('date_time'),
        x='date_time',
        y='traffic_volume',
        title="Traffic Volume Over Time",
        labels={'date_time': 'Date Time', 'traffic_volume': 'Traffic Volume'}
    )

    return fig

# Run the JupyterDash app in inline mode (if running inside a Jupyter Notebook)
if __name__ == '__main__':
    app.run_server(mode='inline', port=8051, debug=True)
```

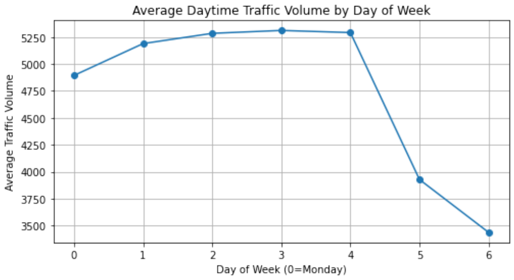
```
Initial Data Snapshot:
  holiday    temp  rain_1h  snow_1h  clouds_all weather_main  \
0     NaN  288.28      0.0      0.0          40       Clouds   
1     NaN  289.36      0.0      0.0          75       Clouds   
2     NaN  289.58      0.0      0.0          90       Clouds   
3     NaN  290.13      0.0      0.0          90       Clouds   
4     NaN  291.14      0.0      0.0          75       Clouds   

  weather_description            date_time  traffic_volume  
0    scattered clouds  2012-10-02 09:00:00            5545  
1       broken clouds  2012-10-02 10:00:00            4516  
2     overcast clouds  2012-10-02 11:00:00            4767  
3     overcast clouds  2012-10-02 12:00:00            5026  
4       broken clouds  2012-10-02 13:00:00            4918  

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48204 entries, 0 to 48203
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   holiday              61 non-null     object 
 1   temp                 48204 non-null  float64
 2   rain_1h              48204 non-null  float64
 3   snow_1h              48204 non-null  float64
 4   clouds_all           48204 non-null  int64  
 5   weather_main         48204 non-null  object 
 6   weather_description  48204 non-null  object 
 7   date_time            48204 non-null  object 
 8   traffic_volume       48204 non-null  int64  
dtypes: float64(3), int64(2), object(4)
memory usage: 3.3+ MB
None

Missing hours in the dataset: 11976
    missing_date_time
0 2012-10-03 07:00:00
1 2012-10-03 10:00:00
2 2012-10-03 11:00:00
3 2012-10-03 17:00:00
4 2012-10-05 02:00:00
```

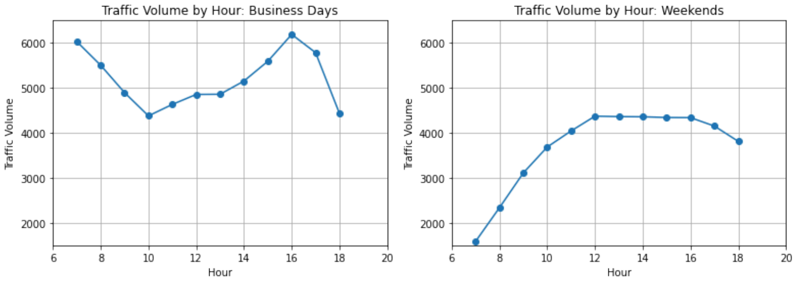


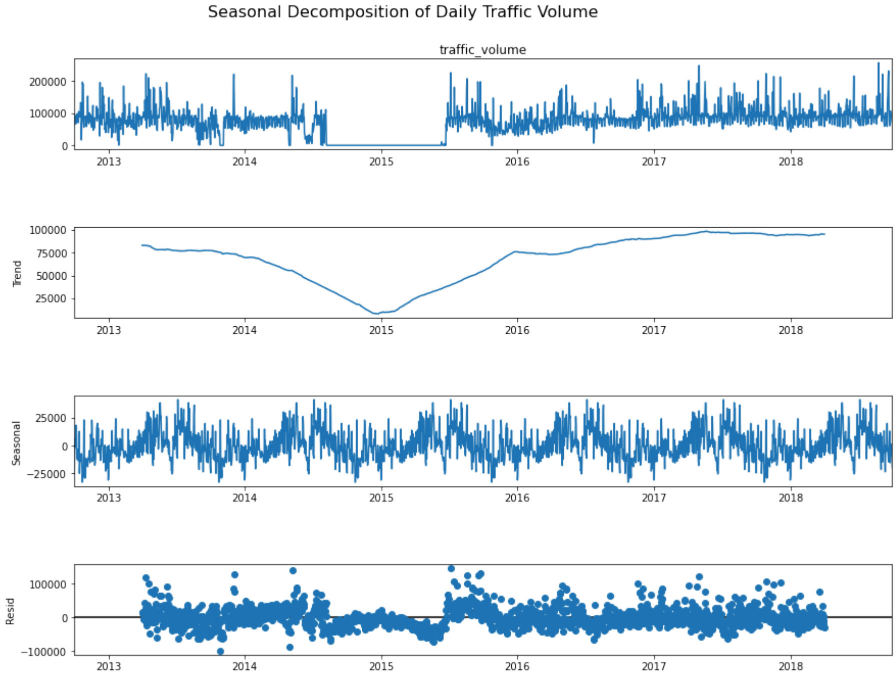
```
Traffic Volume Summary Statistics:
count    48204.000000
mean      3259.818355
std       1986.860670
min          0.000000
25%       1193.000000
50%       3380.000000
75%       4933.000000
max       7280.000000
Name: traffic_volume, dtype: float64

Daytime data shape: (23877, 17)
Nighttime data shape: (24327, 17)
```

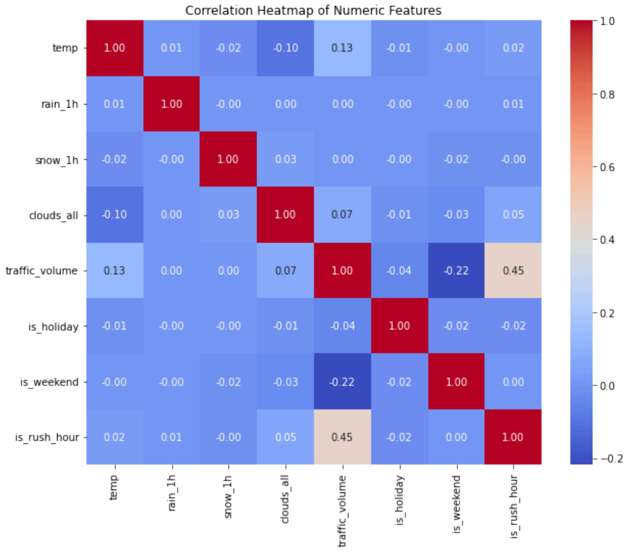




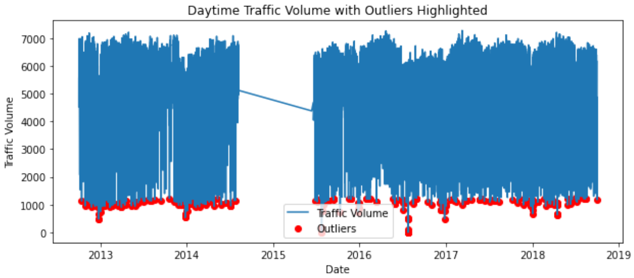


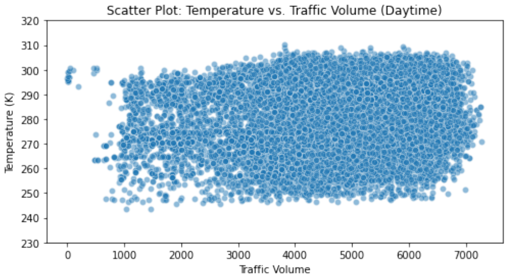


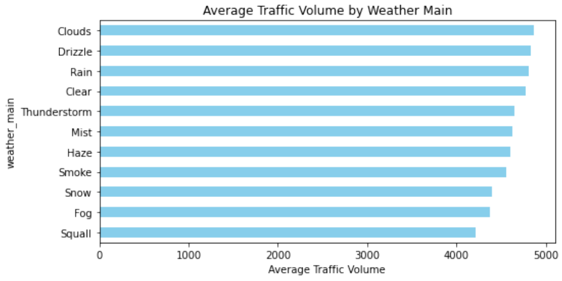


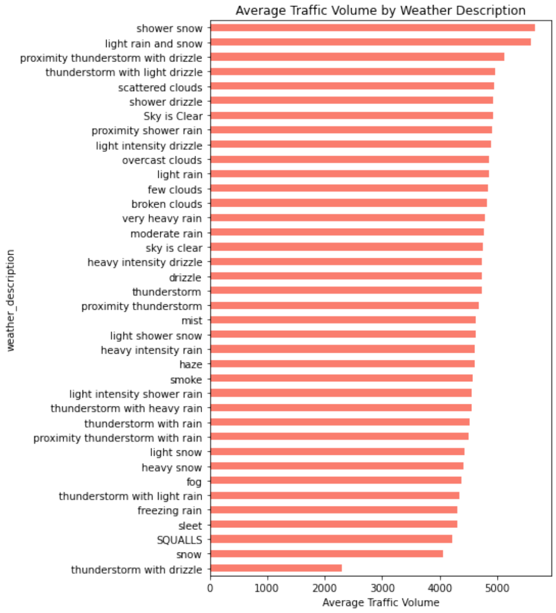


```
Number of outlier hours (z-score > 3) in daytime data: 279
```

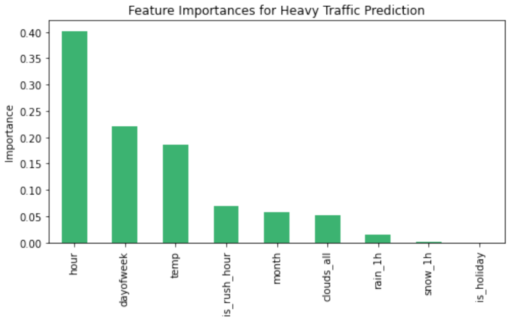




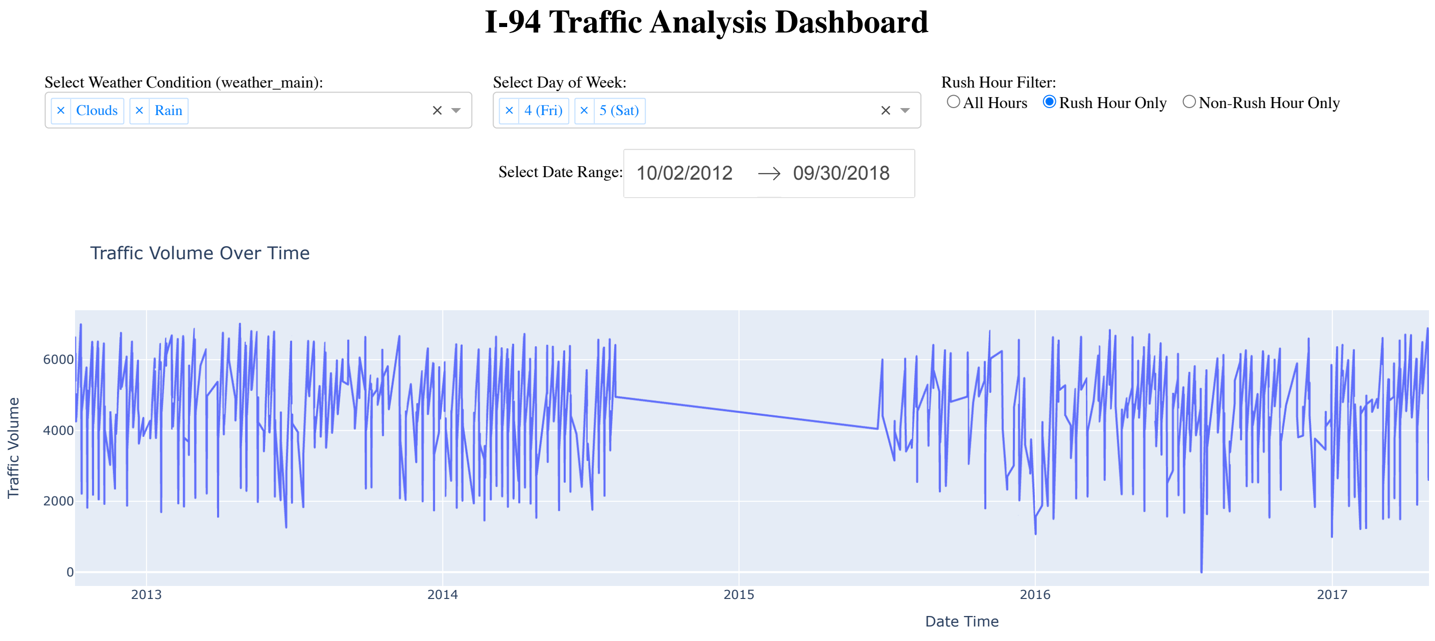




```
Heavy traffic threshold (75th percentile of daytime): 5559 cars/hour

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      5374
           1       0.87      0.86      0.87      1790

    accuracy                           0.93      7164
   macro avg       0.91      0.91      0.91      7164
weighted avg       0.93      0.93      0.93      7164

Confusion Matrix:
[[5153  221]
 [ 248 1542]]
```



```
Dash is running on http://127.0.0.1:8051/
```



I-94 Traffic Analysis Dashboard

Select Weather Condition (weather_main): × Clouds × Rain

Select Day of Week: × 4 (Fri) × 5 (Sat)

Rush Hour Filter: ○ All Hours ● Rush Hour Only ○ Non-Rush Hour Only

Select Date Range: 10/02/2012 → 09/30/2018