In [8]:
```python
import os
os.chdir(r'C:\\Users\\DELL\\Downloads')
```

In [14]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# ------------------------
# 1. Load Raw Data
# ------------------------

df = pd.read_csv('Dataset_Uber Traffic - Dataset_Uber Traffic.csv')

print("Initial Shape:", df.shape)
print(df.head())

# ------------------------
# 2. Clean Data
# ------------------------

# Handle missing values
df = df.dropna()   # or df.fillna(method='ffill') if you prefer imputation
print("After handling missing values:", df.shape)

# Remove duplicates
df = df.drop_duplicates()
print("After removing duplicates:", df.shape)

# Correct data types
df['DateTime'] = pd.to_datetime(df['DateTime'], format="%d/%m/%y %H:%M")  # convert to datetime
df['Junction'] = df['Junction'].astype("category")
df['Vehicles'] = df['Vehicles'].astype(int)

# ------------------------
# 3. Aggregate into hourly intervals per junction
# ------------------------
# (Already seems hourly, but we re-check & aggregate)
df = df.groupby(['Junction', pd.Grouper(key='DateTime', freq='h')])['Vehicles'].sum().reset_index()

print("After Aggregation:", df.shape)
print(df.head())

# ------------------------
# 4. Feature Engineering
# ------------------------
# Extract time features
df['hour'] = df['DateTime'].dt.hour
df['dayofweek'] = df['DateTime'].dt.dayofweek
df['month'] = df['DateTime'].dt.month
df['is_weekend'] = df['dayofweek'].isin([5,6]).astype(int)

# ------------------------
# 5. Normalize Vehicle Counts
# ------------------------
scaler = StandardScaler()
df['Vehicles_scaled'] = scaler.fit_transform(df[['Vehicles']])

print("Final Cleaned Data:")
print(df.head())
```

```
Initial Shape: (48120, 4)
        DateTime  Junction  Vehicles           ID
0  01/11/15 0:00         1        15  20151101001
1  01/11/15 1:00         1        13  20151101011
2  01/11/15 2:00         1        10  20151101021
3  01/11/15 3:00         1         7  20151101031
4  01/11/15 4:00         1         9  20151101041
After handling missing values: (48120, 4)
After removing duplicates: (48120, 4)
After Aggregation: (58368, 3)
  Junction            DateTime  Vehicles
0        1 2015-11-01 00:00:00        15
1        1 2015-11-01 01:00:00        13
2        1 2015-11-01 02:00:00        10
3        1 2015-11-01 03:00:00         7
4        1 2015-11-01 04:00:00         9
Final Cleaned Data:
  Junction            DateTime  Vehicles  hour  dayofweek  month  is_weekend  \
0        1 2015-11-01 00:00:00        15     0          6     11           1
1        1 2015-11-01 01:00:00        13     1          6     11           1
2        1 2015-11-01 02:00:00        10     2          6     11           1
3        1 2015-11-01 03:00:00         7     3          6     11           1
4        1 2015-11-01 04:00:00         9     4          6     11           1

   Vehicles_scaled
0        -0.182725
1        -0.279157
2        -0.423805
3        -0.568452
4        -0.472020
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_50832\3090285124.py:34: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass o
bserved=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  df = df.groupby(['Junction', pd.Grouper(key='DateTime', freq='h')])['Vehicles'].sum().reset_index()
```

In [28]:
```python
# ----------------------------
# Task 3: Data Collection & Integration
# ----------------------------

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# ----------------------------
# 1. Load Traffic Data
# ----------------------------
traffic_df = pd.read_csv("Dataset_Uber Traffic - Dataset_Uber Traffic.csv")

# Convert to datetime
traffic_df['DateTime'] = pd.to_datetime(traffic_df['DateTime'], errors="coerce")

# Remove rows with invalid datetime
traffic_df = traffic_df.dropna(subset=['DateTime'])

# Aggregate into hourly intervals for each junction
traffic_df = traffic_df.groupby(
    ['Junction', pd.Grouper(key='DateTime', freq='h')], observed=True
)['Vehicles'].sum().reset_index()

# ----------------------------
# 2. Load / Collect Weather Data
# ----------------------------
# (In practice: Fetch from OpenWeather API / meteorological sources)
# Here we simulate weather data to show integration

weather_df = pd.DataFrame({
    'DateTime': pd.date_range(start=traffic_df['DateTime'].min(),
                              end=traffic_df['DateTime'].max(),
                              freq="h"),
    'temperature': np.random.uniform(10, 35, size=len(pd.date_range(
        start=traffic_df['DateTime'].min(), end=traffic_df['DateTime'].max(), freq="h"))),
    'precipitation': np.random.uniform(0, 10, size=len(pd.date_range(
        start=traffic_df['DateTime'].min(), end=traffic_df['DateTime'].max(), freq="h"))),
    'humidity': np.random.uniform(30, 90, size=len(pd.date_range(
        start=traffic_df['DateTime'].min(), end=traffic_df['DateTime'].max(), freq="h"))),
    'wind_speed': np.random.uniform(0, 40, size=len(pd.date_range(
        start=traffic_df['DateTime'].min(), end=traffic_df['DateTime'].max(), freq="h")))
})

# ----------------------------
# 3. Load / Collect Event Data
# ----------------------------
# (In practice: Scrape event calendars / government holidays dataset)
# Example: simulated events

event_df = pd.DataFrame({
    'DateTime': pd.date_range(start=traffic_df['DateTime'].min(),
                              end=traffic_df['DateTime'].max(),
                              freq="D"),
    'event': np.random.choice(['None', 'Holiday', 'Concert', 'Sports', 'Protest'],
                              size=len(pd.date_range(start=traffic_df['DateTime'].min(),
                                                     end=traffic_df['DateTime'].max(),
                                                     freq="D")))
})

# ----------------------------
# 4. Merge Traffic + Weather + Events
# ----------------------------
merged_df = pd.merge(traffic_df, weather_df, on="DateTime", how="left")
final_df = pd.merge(merged_df, event_df, on="DateTime", how="left")

# ----------------------------
# 5. Handle Data Quality Issues
# ----------------------------
# Remove duplicates
final_df = final_df.drop_duplicates()

# Handle missing values
numeric_cols = ['temperature', 'precipitation', 'humidity', 'wind_speed', 'Vehicles']
final_df[numeric_cols] = final_df[numeric_cols].fillna(final_df[numeric_cols].mean())

# Normalize / Standardize numeric columns
scaler = StandardScaler()
final_df[numeric_cols] = scaler.fit_transform(final_df[numeric_cols])

# ----------------------------
# 6. Save Final Integrated Dataset
# ----------------------------
final_df.to_csv("Integrated_Traffic_Weather_Event.csv", index=False)

print("✅ Data Collection & Integration Complete!")
print("Final dataset shape:", final_df.shape)
print(final_df.head())
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_50832\3393865936.py:15: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure pars
ing is consistent and as-expected, please specify a format.
  traffic_df['DateTime'] = pd.to_datetime(traffic_df['DateTime'], errors="coerce")
```

```
✅ Data Collection & Integration Complete!
Final dataset shape: (48120, 8)
   Junction            DateTime  Vehicles  temperature  precipitation  \
0         1 2015-01-11 00:00:00 -0.375489     0.357391       0.237071
1         1 2015-01-11 01:00:00 -0.471875    -0.733105      -0.251813
2         1 2015-01-11 02:00:00 -0.616454     0.076729       0.282657
3         1 2015-01-11 03:00:00 -0.761034    -0.668178       0.844581
4         1 2015-01-11 04:00:00 -0.664648     0.239116      -1.644648

   humidity  wind_speed   event
0 -0.366372   -0.041753  Sports
1  1.609505   -1.638922     NaN
2 -0.252250   -0.774879     NaN
3  1.425694    0.865230     NaN
4 -0.017557    0.896137     NaN
```

In [30]:
```python
##Task 4: Model Development, Training, and Refinement
from sklearn.model_selection import TimeSeriesSplit

# Sort dataset by datetime
final_df = final_df.sort_values("DateTime")

# Features and target
X = final_df[['temperature', 'precipitation', 'humidity', 'wind_speed']]  # features
y = final_df['Vehicles']  # target

# Time-based split (e.g., last 20% data for validation)
split_index = int(len(final_df) * 0.8)
X_train, X_val = X.iloc[:split_index], X.iloc[split_index:]
y_train, y_val = y.iloc[:split_index], y.iloc[split_index:]
```

In [35]:
```
pip install xgboost
```

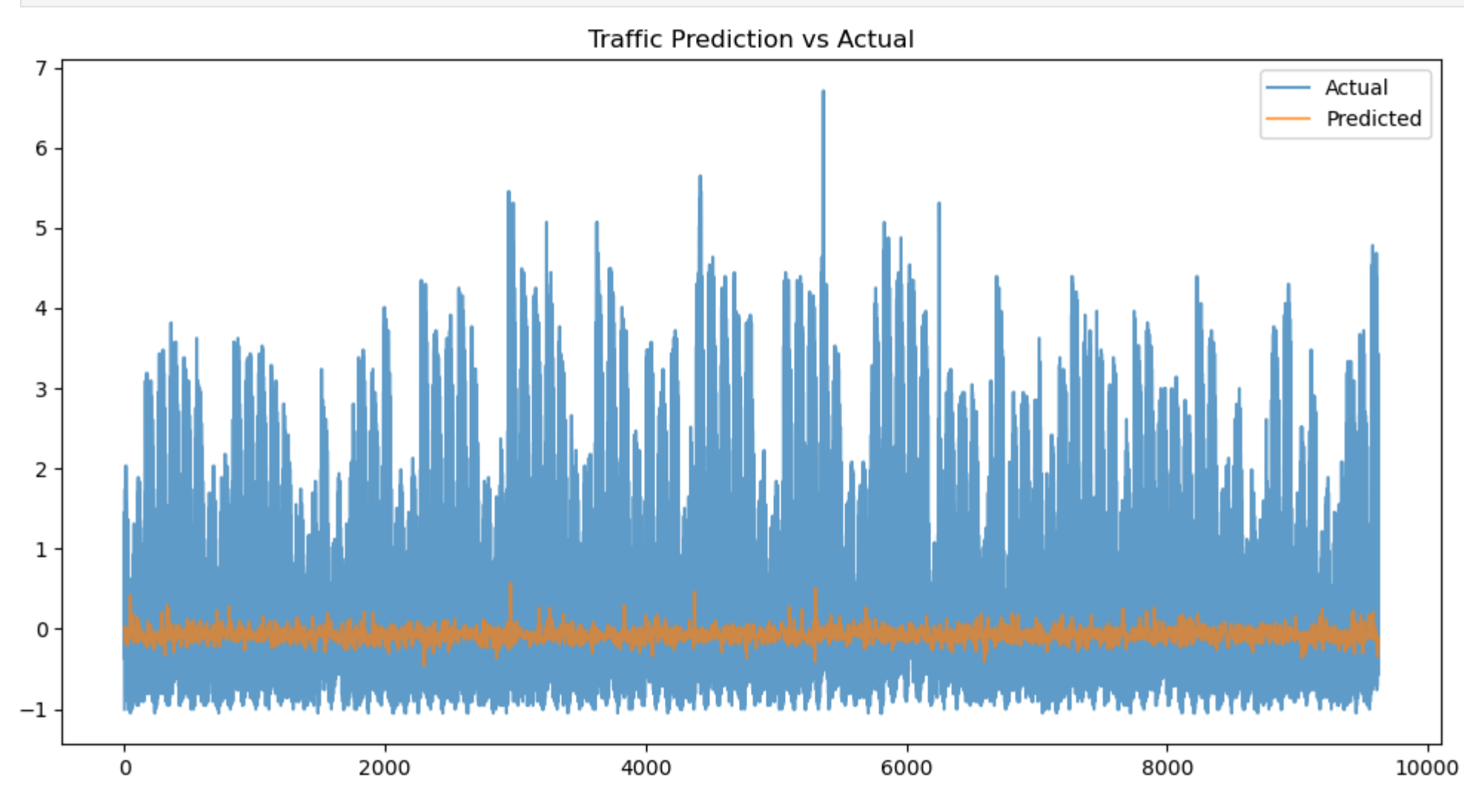
```
Requirement already satisfied: xgboost in c:\programdata\anaconda3\lib\site-packages (3.0.5)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.13.1)
Note: you may need to restart the kernel to use updated packages.
```

In [37]:
```python
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Train XGBoost
model = xgb.XGBRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_val)
```

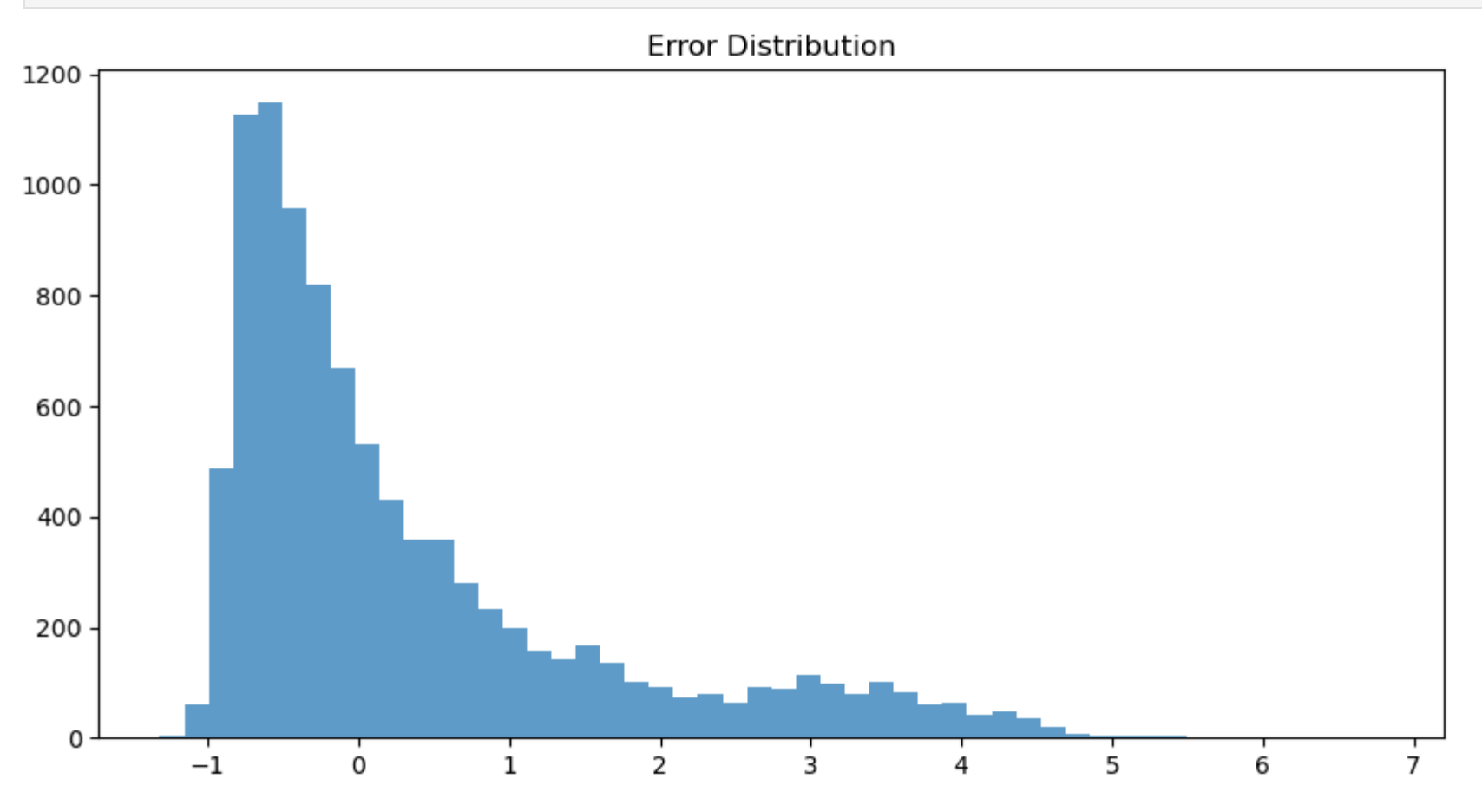
In [39]:
```python
##Step 2: Model Evaluation & Cross Validation
mae = mean_absolute_error(y_val, y_pred)
rmse = np.sqrt(mean_squared_error(y_val, y_pred))
r2 = r2_score(y_val, y_pred)

print("MAE:", mae)
print("RMSE:", rmse)
print("R²:", r2)
```

```
MAE: 0.9167345556808822
RMSE: 1.3432569507496668
R²: -0.07644234210637646
```

In [41]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.plot(y_val.values, label="Actual", alpha=0.7)
plt.plot(y_pred, label="Predicted", alpha=0.7)
plt.legend()
plt.title("Traffic Prediction vs Actual")
plt.show()
```



In [43]:
```python
residuals = y_val - y_pred
plt.figure(figsize=(10,5))
plt.hist(residuals, bins=50, alpha=0.7)
plt.title("Error Distribution")
plt.show()
```



In [45]:
```python
##3. Cross-Validation (Time-Series Aware)
tscv = TimeSeriesSplit(n_splits=5)
cv_results = []

for train_idx, test_idx in tscv.split(X):
    X_tr, X_te = X.iloc[train_idx], X.iloc[test_idx]
    y_tr, y_te = y.iloc[train_idx], y.iloc[test_idx]

    model.fit(X_tr, y_tr)
    y_hat = model.predict(X_te)

    mae = mean_absolute_error(y_te, y_hat)
    rmse = np.sqrt(mean_squared_error(y_te, y_hat))
    r2 = r2_score(y_te, y_hat)

    cv_results.append((mae, rmse, r2))

print("Cross-validation results:")
for i, (mae, rmse, r2) in enumerate(cv_results, 1):
    print(f"Fold {i}: MAE={mae:.3f}, RMSE={rmse:.3f}, R²={r2:.3f}")
```

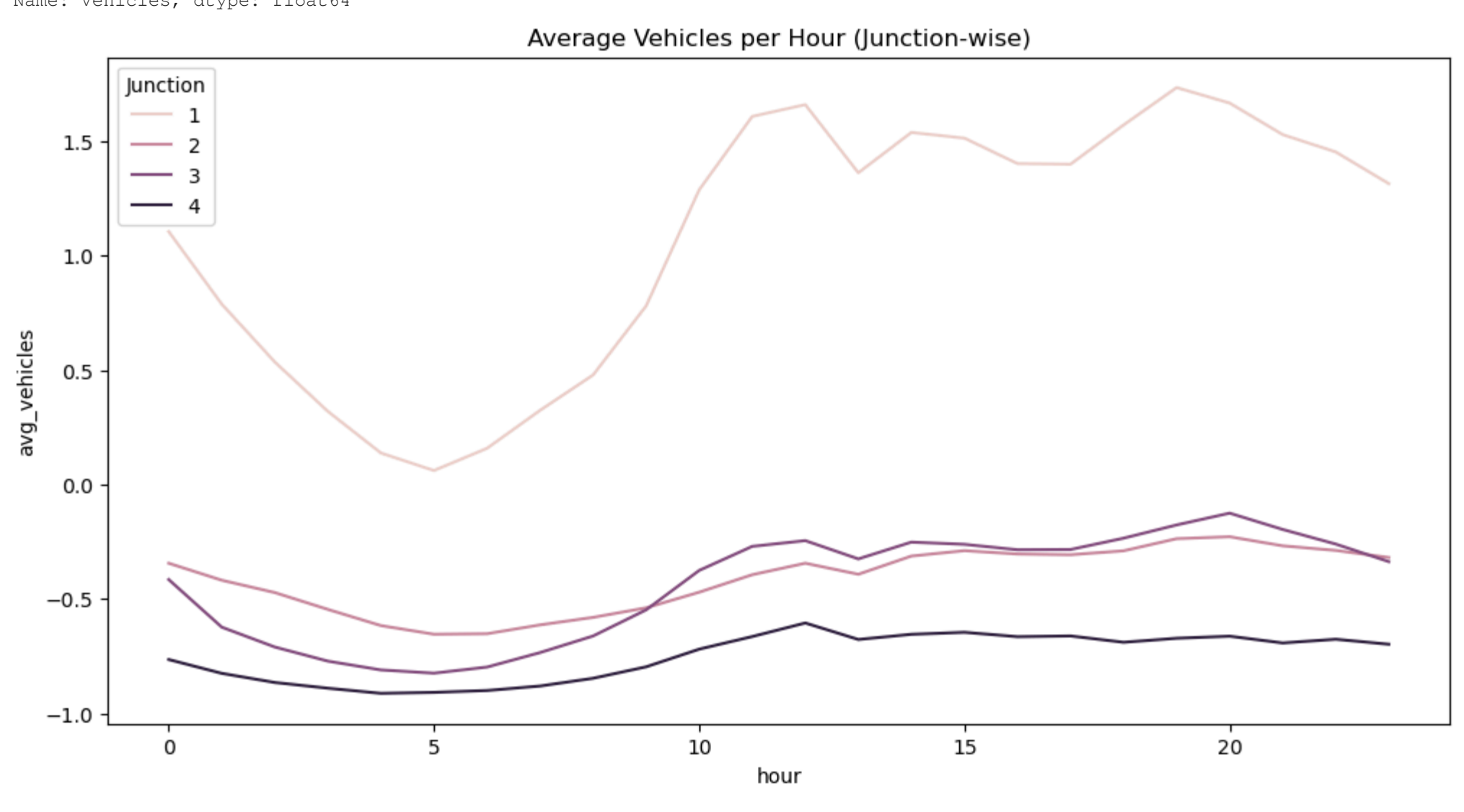
```
Cross-validation results:
Fold 1: MAE=0.507, RMSE=0.731, R²=-0.142
Fold 2: MAE=0.624, RMSE=0.914, R²=-0.111
Fold 3: MAE=0.706, RMSE=1.033, R²=-0.091
Fold 4: MAE=0.849, RMSE=1.233, R²=-0.091
Fold 5: MAE=0.931, RMSE=1.369, R²=-0.070
```

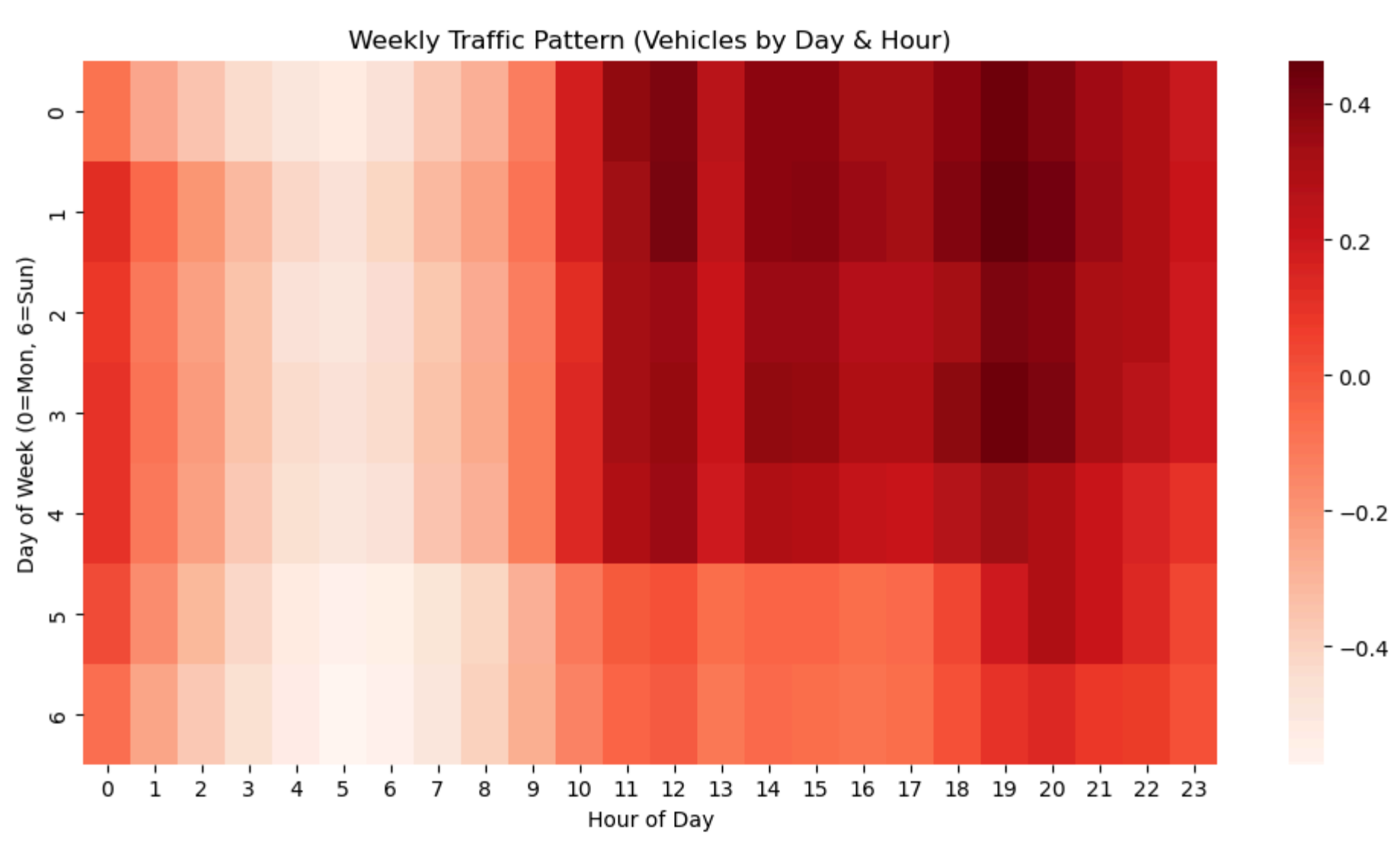
In [47]:
```python
##Step 3: Model Refinement
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    "n_estimators": [100, 200, 500],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1, 0.2],
    "subsample": [0.7, 0.8, 1.0],
}

search = RandomizedSearchCV(
    xgb.XGBRegressor(random_state=42),
    param_distributions=param_dist,
    n_iter=10,
    scoring="neg_mean_absolute_error",
    cv=3,
    random_state=42,
    n_jobs=-1
)

search.fit(X_train, y_train)
print("Best params:", search.best_params_)
```

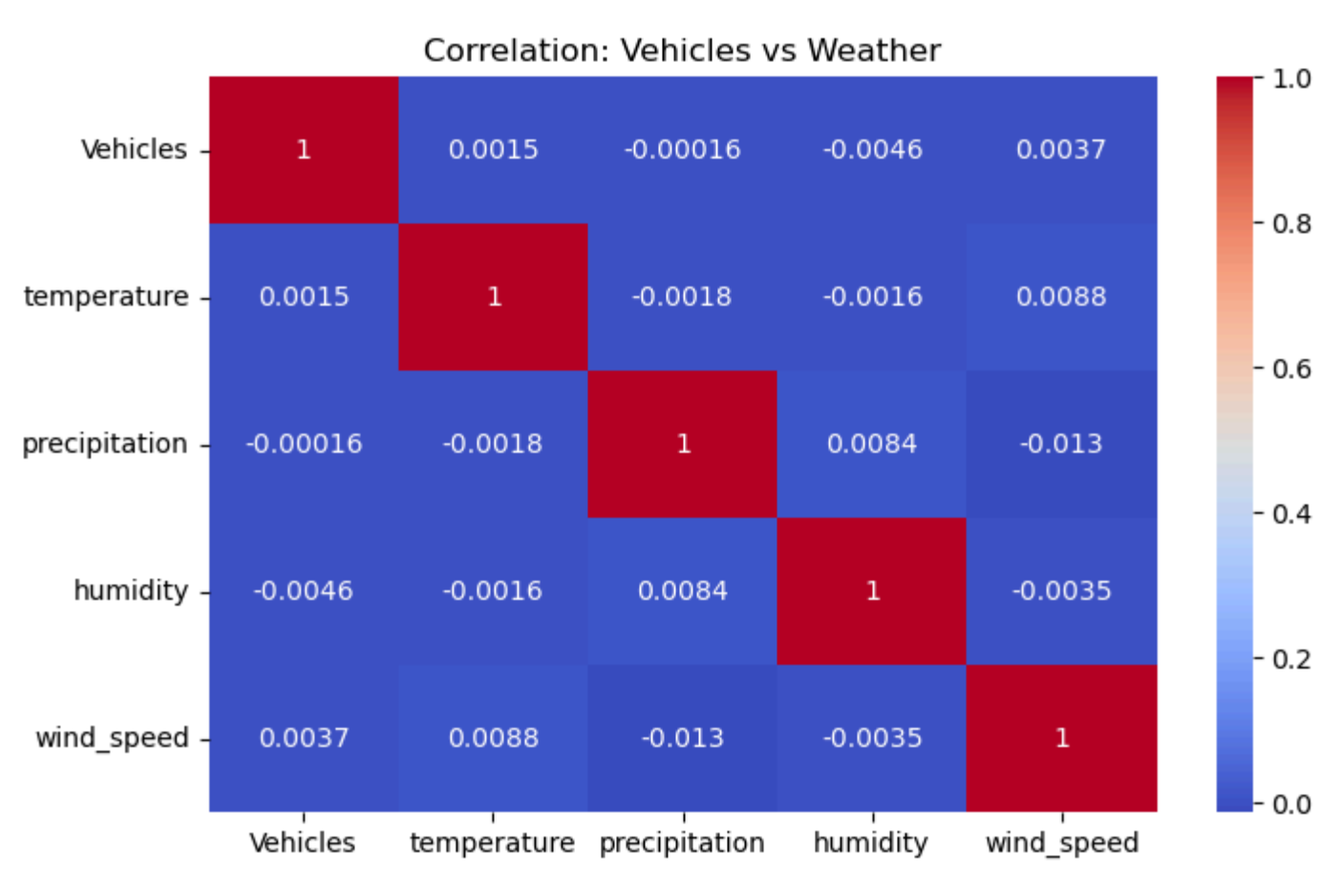
```
Best params: {'subsample': 0.7, 'n_estimators': 100, 'max_depth': 7, 'learning_rate': 0.01}
```

In [51]:
```python
# ----------------------------
# Task 5: Peak Hour Identification & Pattern Analysis
# ----------------------------

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Make sure final_df from Task 3/4 is available
# final_df has ['DateTime','Junction','Vehicles','temperature','precipitation','humidity','wind_speed','event']

# ----------------------------
# 1. Congestion Metrics
# ----------------------------
# Extract hour, weekday, and month for analysis
final_df['hour'] = final_df['DateTime'].dt.hour
final_df['day_of_week'] = final_df['DateTime'].dt.dayofweek  # 0=Monday, 6=Sunday
final_df['month'] = final_df['DateTime'].dt.month

# Calculate average traffic per hour per junction
hourly_congestion = final_df.groupby(['Junction','hour'])['Vehicles'].mean().reset_index()
hourly_congestion.rename(columns={'Vehicles':'avg_vehicles'}, inplace=True)

# ----------------------------
# 2. Identify Peak Hours
# ----------------------------
# Find top 3 peak hours per junction
peak_hours = hourly_congestion.sort_values(['Junction','avg_vehicles'], ascending=[True,False]) \
                              .groupby('Junction').head(3)

print("🚦 Peak Hours per Junction:")
print(peak_hours)

# ----------------------------
# 3. Temporal Patterns
# ----------------------------
# Avg vehicles by day of week
weekly_pattern = final_df.groupby(['day_of_week','hour'])['Vehicles'].mean().reset_index()

# Avg vehicles by month
monthly_pattern = final_df.groupby(['month','hour'])['Vehicles'].mean().reset_index()

# ----------------------------
# 4. Influencing Factors
# ----------------------------
# Correlation between traffic and weather
weather_corr = final_df[['Vehicles','temperature','precipitation','humidity','wind_speed']].corr()

print("\n🌦️ Correlation between Traffic & Weather:")
print(weather_corr['Vehicles'])

# ----------------------------
# Visualizations
# ----------------------------
plt.figure(figsize=(12,6))
sns.lineplot(data=hourly_congestion, x='hour', y='avg_vehicles', hue='Junction')
plt.title("Average Vehicles per Hour (Junction-wise)")
plt.show()

# ----------------------------
# Weekly Traffic Heatmap
# ----------------------------
plt.figure(figsize=(12,6))
sns.heatmap(
    weekly_pattern.pivot(index='day_of_week', columns='hour', values='Vehicles'),
    cmap='Reds'
)
plt.title("Weekly Traffic Pattern (Vehicles by Day & Hour)")
plt.ylabel("Day of Week (0=Mon, 6=Sun)")
plt.xlabel("Hour of Day")
plt.show()


plt.figure(figsize=(8,5))
sns.heatmap(weather_corr, annot=True, cmap='coolwarm')
plt.title("Correlation: Vehicles vs Weather")
plt.show()
```

```
🚦 Peak Hours per Junction:
    Junction  hour  avg_vehicles
19         1    19      1.735576
20         1    20      1.667963
12         1    12      1.660909
44         2    20     -0.227977
43         2    19     -0.236537
45         2    21     -0.267688
68         3    20     -0.124853
67         3    19     -0.176454
69         3    21     -0.196112
84         4    12     -0.605005
87         4    15     -0.645743
86         4    14     -0.654530

🌦 Correlation between Traffic & Weather:
Vehicles         1.000000
temperature      0.001474
precipitation   -0.000164
humidity        -0.004553
wind_speed       0.003728
Name: Vehicles, dtype: float64
```







In [ ]: