Group_Project_2

March 9, 2025

[75]:
```
# Importing dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.tsa.api import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statsmodels.api as sm
```

[55]:
```
# Read in the data
df = pd.read_csv('/Users/helenamabey/Stats_Spring_2025/Congestion.csv')
df.head()
```

[55]:
```
      Date  Year Month  Daily Vehicle Count  Regular Gas Price Average  \
0  2005-01  2005   Jan                19470                     1.7660
1  2005-02  2005   Feb                21207                     1.8550
2  2005-03  2005   Mar                22943                     2.0825
3  2005-04  2005   Apr                21288                     2.2300
4  2005-05  2005   May                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[57]:

```
# Review the data, confirm no nulls and all data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146 entries, 0 to 145
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Date                         146 non-null    object
 1   Year                         146 non-null    int64
 2   Month                        146 non-null    object
 3   Daily Vehicle Count          146 non-null    int64
 4   Regular Gas Price Average    146 non-null    float64
 5   Midgrade Gas Price Average   146 non-null    float64
 6   Premium Gas Price Average    146 non-null    float64
 7   Monthly_Max_Temp             146 non-null    float64
 8   Monthly_Min_Temp             146 non-null    float64
 9   Monthly_Sum_Precipitation    146 non-null    float64
 10  Monthly_Snowfall             146 non-null    float64
dtypes: float64(7), int64(2), object(2)
memory usage: 12.7+ KB
```

[59]:

```
# Convert Date to datatime data type
df['Date'] = pd.to_datetime(df['Date'])
df.head()
```

[59]:

```
        Date  Year Month  Daily Vehicle Count  Regular Gas Price Average  \
0 2005-01-01  2005   Jan                19470                     1.7660
1 2005-02-01  2005   Feb                21207                     1.8550
2 2005-03-01  2005   Mar                22943                     2.0825
3 2005-04-01  2005   Apr                21288                     2.2300
4 2005-05-01  2005   May                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[61]:
```
# Convert Month and Year to Month-Year format (ChatGPT)
df['Month Year'] = df['Date'].dt.to_period('M')

df.head()
```

[61]:
```
        Date  Year Month  Daily Vehicle Count  Regular Gas Price Average  \
0 2005-01-01  2005   Jan                19470                     1.7660
1 2005-02-01  2005   Feb                21207                     1.8550
2 2005-03-01  2005   Mar                22943                     2.0825
3 2005-04-01  2005   Apr                21288                     2.2300
4 2005-05-01  2005   May                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall Month Year
0              12.7                       2.29              36.0    2005-01
1              10.5                       2.14              37.5    2005-02
2              17.5                       1.49              20.0    2005-03
3              24.0                       2.24              20.0    2005-04
4              30.5                       1.01               2.0    2005-05
```

[63]:
```
# Check out the data again to confirm data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146 entries, 0 to 145
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        146 non-null    datetime64[ns]
 1   Year                        146 non-null    int64
 2   Month                       146 non-null    object
 3   Daily Vehicle Count         146 non-null    int64
 4   Regular Gas Price Average   146 non-null    float64
 5   Midgrade Gas Price Average  146 non-null    float64
 6   Premium Gas Price Average   146 non-null    float64
 7   Monthly_Max_Temp            146 non-null    float64
 8   Monthly_Min_Temp            146 non-null    float64
 9   Monthly_Sum_Precipitation   146 non-null    float64
 10  Monthly_Snowfall            146 non-null    float64
 11  Month Year                  146 non-null    period[M]
dtypes: datetime64[ns](1), float64(7), int64(2), object(1), period[M](1)
```

```
memory usage: 13.8+ KB
```

[65]:
```python
# Reorder columns to move Month Year to the first column
df = df[['Month Year'] + [col for col in df.columns if col != 'Month Year']]

df.head()
```

[65]:
```
  Month Year       Date  Year Month  Daily Vehicle Count  \
0    2005-01 2005-01-01  2005   Jan                19470
1    2005-02 2005-02-01  2005   Feb                21207
2    2005-03 2005-03-01  2005   Mar                22943
3    2005-04 2005-04-01  2005   Apr                21288
4    2005-05 2005-05-01  2005   May                23505

   Regular Gas Price Average  Midgrade Gas Price Average  \
0                     1.7660                      1.8760
1                     1.8550                      1.9650
2                     2.0825                      2.1900
3                     2.2300                      2.3425
4                     2.1540                      2.2640

   Premium Gas Price Average  Monthly_Max_Temp  Monthly_Min_Temp  \
0                     1.9800              32.3              12.7
1                     2.0650              33.9              10.5
2                     2.2875              39.5              17.5
3                     2.4450              49.9              24.0
4                     2.3640              62.1              30.5

   Monthly_Sum_Precipitation  Monthly_Snowfall
0                       2.29              36.0
1                       2.14              37.5
2                       1.49              20.0
3                       2.24              20.0
4                       1.01               2.0
```

[69]:
```python
# Remove month and year fields prior to correlation review
df.pop('Year')
df.pop('Month')
df.head()
```

[69]:
```
  Month Year       Date  Daily Vehicle Count  Regular Gas Price Average  \
0    2005-01 2005-01-01                19470                     1.7660
1    2005-02 2005-02-01                21207                     1.8550
2    2005-03 2005-03-01                22943                     2.0825
3    2005-04 2005-04-01                21288                     2.2300
4    2005-05 2005-05-01                23505                     2.1540
```

```
   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[71]:

```
# Check the data for obvious trends. Snowfall is showing a strong right-skew␣
↪based on mean and median values
df.describe()
```

[71]:

```
       Daily Vehicle Count  Regular Gas Price Average  \
count           146.000000                 146.000000
mean          23604.958904                   2.798003
std            3392.757418                   0.612065
min           18208.000000                   1.590000
25%           20696.500000                   2.232000
50%           22957.000000                   2.770000
75%           26287.750000                   3.392500
max           33354.000000                   3.995000

       Midgrade Gas Price Average  Premium Gas Price Average  \
count                  146.000000                 146.000000
mean                     2.937860                   3.070890
std                      0.596528                   0.585927
min                      1.702500                   1.812500
25%                      2.437500                   2.648125
50%                      2.891500                   3.003750
75%                      3.510625                   3.635000
max                      4.112500                   4.225000

       Monthly_Max_Temp  Monthly_Min_Temp  Monthly_Sum_Precipitation  \
count        146.000000        146.000000                 146.000000
mean          51.568493         23.796575                   1.862877
std           17.499666         12.925380                   0.881182
min           23.600000          1.200000                   0.170000
25%           36.000000         12.625000                   1.222500
50%           51.200000         24.450000                   1.810000
75%           68.525000         35.175000                   2.427500
max           79.700000         45.200000                   3.970000
```

```
       Monthly_Snowfall
count        146.000000
mean          13.247945
std           15.851976
min            0.000000
25%            0.000000
50%            6.750000
75%           21.400000
max           57.000000
```

[73]:
```
# Review correlation. Min and Max temp show a strong positive correlation.␣
↪Snowfall is moderately correlated.
# Still would like to test some gas price values in the model
df.corr()
```

```
/var/folders/tb/9ztd83zd25xb9w5xfq415x100000gn/T/ipykernel_36812/1134722465.py:1
: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  df.corr()
```

[73]:
```
                            Daily Vehicle Count  Regular Gas Price Average  \
Daily Vehicle Count                    1.000000                   0.116771
Regular Gas Price Average              0.116771                   1.000000
Midgrade Gas Price Average             0.146964                   0.997028
Premium Gas Price Average              0.175799                   0.988014
Monthly_Max_Temp                       0.823199                   0.359248
Monthly_Min_Temp                       0.819662                   0.334312
Monthly_Sum_Precipitation             -0.034294                  -0.013109
Monthly_Snowfall                      -0.583019                  -0.243885

                            Midgrade Gas Price Average  \
Daily Vehicle Count                           0.146964
Regular Gas Price Average                     0.997028
Midgrade Gas Price Average                    1.000000
Premium Gas Price Average                     0.996939
Monthly_Max_Temp                              0.367900
Monthly_Min_Temp                              0.344863
Monthly_Sum_Precipitation                    -0.013135
Monthly_Snowfall                             -0.244603

                            Premium Gas Price Average  Monthly_Max_Temp  \
Daily Vehicle Count                          0.175799          0.823199
Regular Gas Price Average                    0.988014          0.359248
Midgrade Gas Price Average                   0.996939          0.367900
Premium Gas Price Average                    1.000000          0.373319
```

```
Monthly_Max_Temp                           0.373319          1.000000
Monthly_Min_Temp                           0.352280          0.984329
Monthly_Sum_Precipitation                 -0.012910         -0.061773
Monthly_Snowfall                          -0.242776         -0.747067

                           Monthly_Min_Temp  Monthly_Sum_Precipitation  \
Daily Vehicle Count                0.819662                  -0.034294
Regular Gas Price Average          0.334312                  -0.013109
Midgrade Gas Price Average         0.344863                  -0.013135
Premium Gas Price Average          0.352280                  -0.012910
Monthly_Max_Temp                   0.984329                  -0.061773
Monthly_Min_Temp                   1.000000                   0.020042
Monthly_Sum_Precipitation          0.020042                   1.000000
Monthly_Snowfall                  -0.724182                   0.384635

                           Monthly_Snowfall
Daily Vehicle Count               -0.583019
Regular Gas Price Average         -0.243885
Midgrade Gas Price Average        -0.244603
Premium Gas Price Average         -0.242776
Monthly_Max_Temp                  -0.747067
Monthly_Min_Temp                  -0.724182
Monthly_Sum_Precipitation          0.384635
Monthly_Snowfall                   1.000000
```

[229]:

```python
correlation_max = df[['Daily Vehicle Count', 'Monthly_Max_Temp']].corr()
correlation_max
```

[229]:

	Daily Vehicle Count	Monthly_Max_Temp
Daily Vehicle Count	1.000000	0.823199
Monthly_Max_Temp	0.823199	1.000000

[231]:

```python
correlation_min = df[['Daily Vehicle Count', 'Monthly_Min_Temp']].corr()
correlation_min
```

[231]:

	Daily Vehicle Count	Monthly_Min_Temp
Daily Vehicle Count	1.000000	0.819662
Monthly_Min_Temp	0.819662	1.000000

[233]:

```python
correlation_gas = df[['Daily Vehicle Count', 'Regular Gas Price Average']].
    ↪corr()
correlation_gas
```

[233]:

	Daily Vehicle Count	Regular Gas Price Average
Daily Vehicle Count	1.000000	0.116771
Regular Gas Price Average	0.116771	1.000000

[77]:
```python
# Set Date as index for modeling and visualizations
df.set_index('Date', inplace=True)  # Set it as time series index
df.head()
```

[77]:
```
           Month Year  Daily Vehicle Count  Regular Gas Price Average  \
Date
2005-01-01    2005-01                19470                     1.7660
2005-02-01    2005-02                21207                     1.8550
2005-03-01    2005-03                22943                     2.0825
2005-04-01    2005-04                21288                     2.2300
2005-05-01    2005-05                23505                     2.1540

            Midgrade Gas Price Average  Premium Gas Price Average  \
Date
2005-01-01                      1.8760                     1.9800
2005-02-01                      1.9650                     2.0650
2005-03-01                      2.1900                     2.2875
2005-04-01                      2.3425                     2.4450
2005-05-01                      2.2640                     2.3640

            Monthly_Max_Temp  Monthly_Min_Temp  Monthly_Sum_Precipitation  \
Date
2005-01-01              32.3              12.7                       2.29
2005-02-01              33.9              10.5                       2.14
2005-03-01              39.5              17.5                       1.49
2005-04-01              49.9              24.0                       2.24
2005-05-01              62.1              30.5                       1.01

            Monthly_Snowfall
Date
2005-01-01              36.0
2005-02-01              37.5
2005-03-01              20.0
2005-04-01              20.0
2005-05-01               2.0
```

[79]:
```python
# Select features (independent variables) and dependent variable
X = df[['Regular Gas Price Average', 'Midgrade Gas Price Average', 'Premium Gas␣
 ↪Price Average',
        'Monthly_Max_Temp', 'Monthly_Min_Temp', 'Monthly_Sum_Precipitation',␣
 ↪'Monthly_Snowfall']]

y = df['Daily Vehicle Count']  # Dependent variable
```

[81]:
```python
# Train-test split (train up to 2016, test 2017)
train_mask = df.index.year < 2017
X_train, X_test = X[train_mask], X[~train_mask]
```

```
y_train, y_test = y[train_mask], y[~train_mask]
```

[83]:
```
# Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

[83]:
```
LinearRegression()
```

[85]:
```
# Predict on test set, 2017-split between years, not a good indicator. Will try␣
↪70/30 split
y_pred = model.predict(X_test)
```

[87]:
```
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Model Performance:\n MAE: {mae:.2f}\n R² Score: {r2:.4f}")
```

```
Model Performance:
 MAE: 1245.69
 R² Score: -1.8582
```

[89]:
```
# Fit the model using statsmodels
model_sm = sm.OLS(y_train, X_train).fit()
```

[91]:
```
# Model summary
model_summary = model_sm.summary()
print(model_summary)
# Very overfit with multicollearity relationships between the independent␣
↪variables
```

```
                                  OLS Regression Results
================================================================================
========
Dep. Variable:     Daily Vehicle Count   R-squared (uncentered):
0.994
Model:                             OLS   Adj. R-squared (uncentered):
0.993
Method:                  Least Squares   F-statistic:
3052.
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):
4.81e-147
Time:                         19:16:10   Log-Likelihood:
-1291.8
No. Observations:                  144   AIC:
2598.
Df Residuals:                      137   BIC:
```

```
2618.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
==============
                              coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
--------------
Regular Gas Price Average  -3.819e+04   2.43e+04     -1.575      0.118
-8.62e+04    9764.846
Midgrade Gas Price Average  4.239e+04   4.93e+04      0.859      0.392
-5.52e+04     1.4e+05
Premium Gas Price Average  -3780.2288   2.52e+04     -0.150      0.881
-5.36e+04     4.6e+04
Monthly_Max_Temp             507.9830     40.803     12.450      0.000
427.298     588.668
Monthly_Min_Temp            -436.2436     60.133     -7.255      0.000
-555.153    -317.335
Monthly_Sum_Precipitation    703.4290    231.062      3.044      0.003
246.520    1160.337
Monthly_Snowfall              26.1910     18.461      1.419      0.158
-10.315      62.696
==============================================================================
Omnibus:                       12.140   Durbin-Watson:                   1.354
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               15.542
Skew:                          -0.516   Prob(JB):                     0.000422
Kurtosis:                       4.235   Cond. No.                     2.29e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 2.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[93]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Compute VIF for each independent variable
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]

print(vif_data)
```

```
# Proves the multicollinearity in this model
```

```
                      Feature            VIF
0   Regular Gas Price Average  183905.254521
1  Midgrade Gas Price Average  833512.715877
2   Premium Gas Price Average  235561.609059
3            Monthly_Max_Temp     180.968225
4            Monthly_Min_Temp      97.531866
5   Monthly_Sum_Precipitation       8.562524
6            Monthly_Snowfall       5.428794
```

[101]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Add a constant column for VIF calculation
X_vif = sm.add_constant(X)

# Compute VIF for each independent variable
vif_data = pd.DataFrame()
vif_data["Feature"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in␣
 ↪range(X_vif.shape[1])]

print(vif_data)
# Even adding in the constant, this model is not successful
```

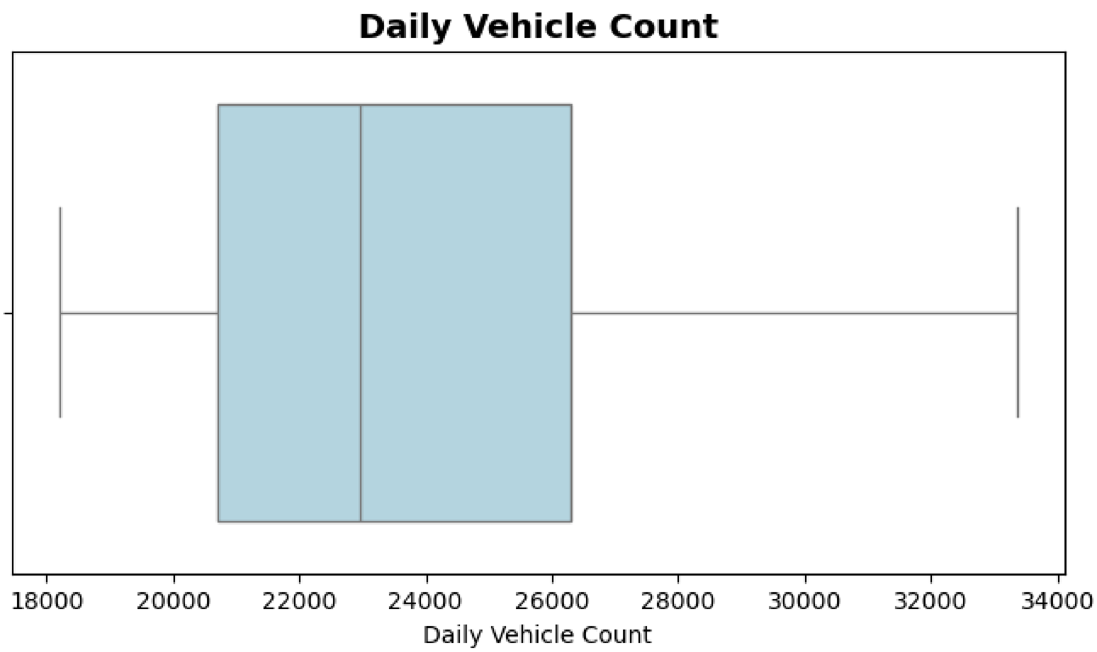
```
                      Feature           VIF
0                       const    124.316845
1   Regular Gas Price Average   8603.709217
2  Midgrade Gas Price Average  33465.881552
3   Premium Gas Price Average   8313.528987
4            Monthly_Max_Temp     43.278251
5            Monthly_Min_Temp     44.468607
6   Monthly_Sum_Precipitation      1.804832
7            Monthly_Snowfall      3.254614
```

[105]:

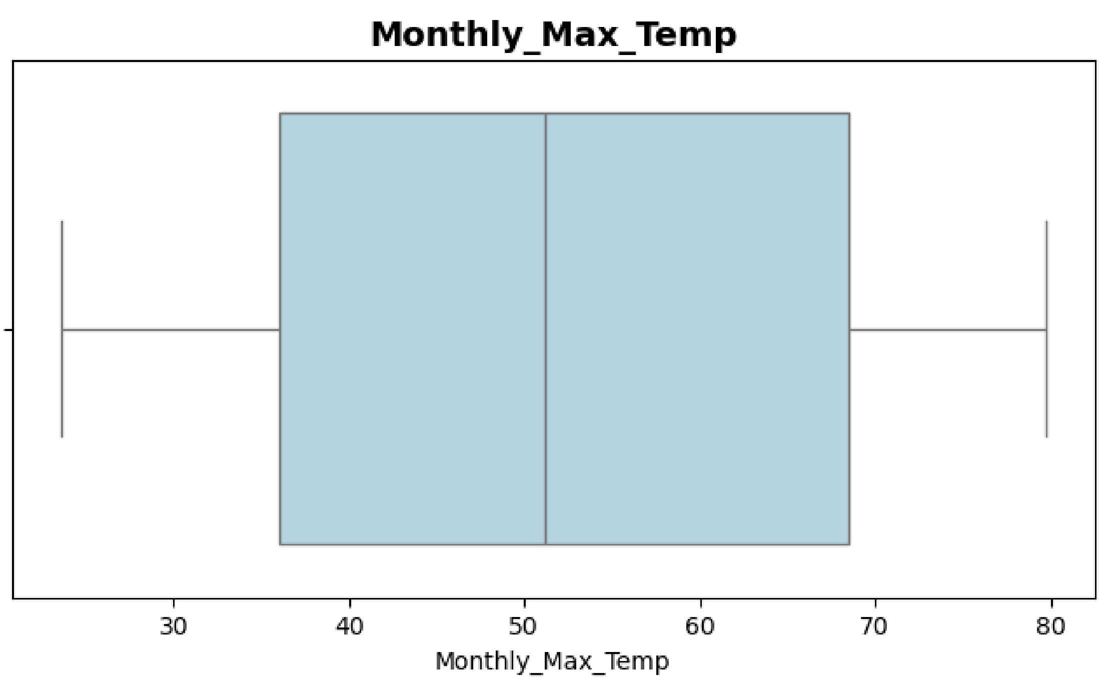
```
df.to_csv('/Users/helenamabey/Stats_Spring_2025/Congestion Cleaned.csv')
```

[109]:

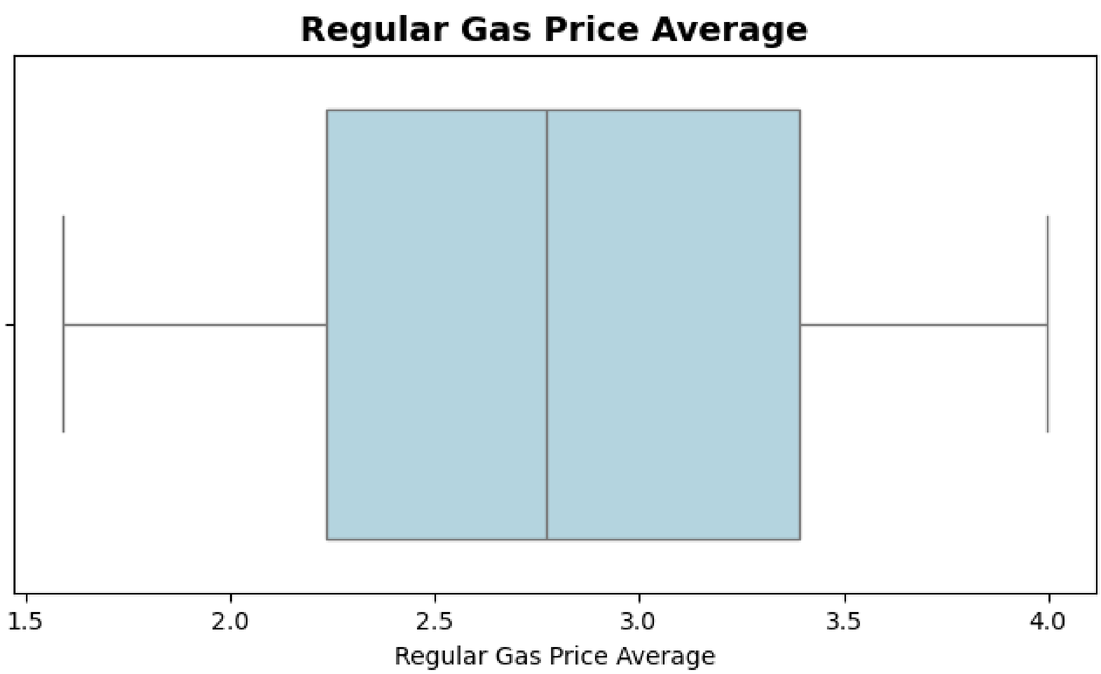
```
import seaborn as sns
# Boxplot for Daily Vehicle Count. Slight right-skew for the count data, could␣
 ↪indicate some outliers
plt.figure(figsize=(8, 4))
sns.boxplot(x=df["Daily Vehicle Count"], color="lightblue")
plt.title("Daily Vehicle Count", fontsize=14, fontweight='bold')
plt.show()
```



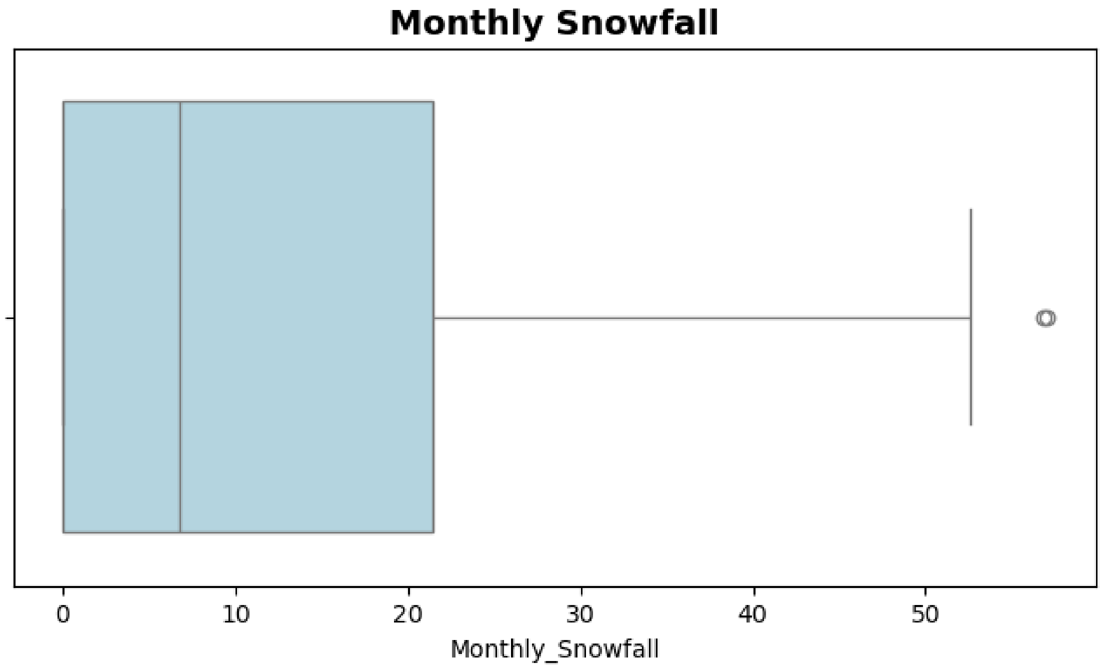
[111]:
```
# Boxplot for Monthly_Max_Temp. This distribution is nearly normal with a very
↪slight right-skew
plt.figure(figsize=(8, 4))
sns.boxplot(x=df["Monthly_Max_Temp"], color="lightblue")
plt.title("Monthly_Max_Temp", fontsize=14, fontweight='bold')
plt.show()
```



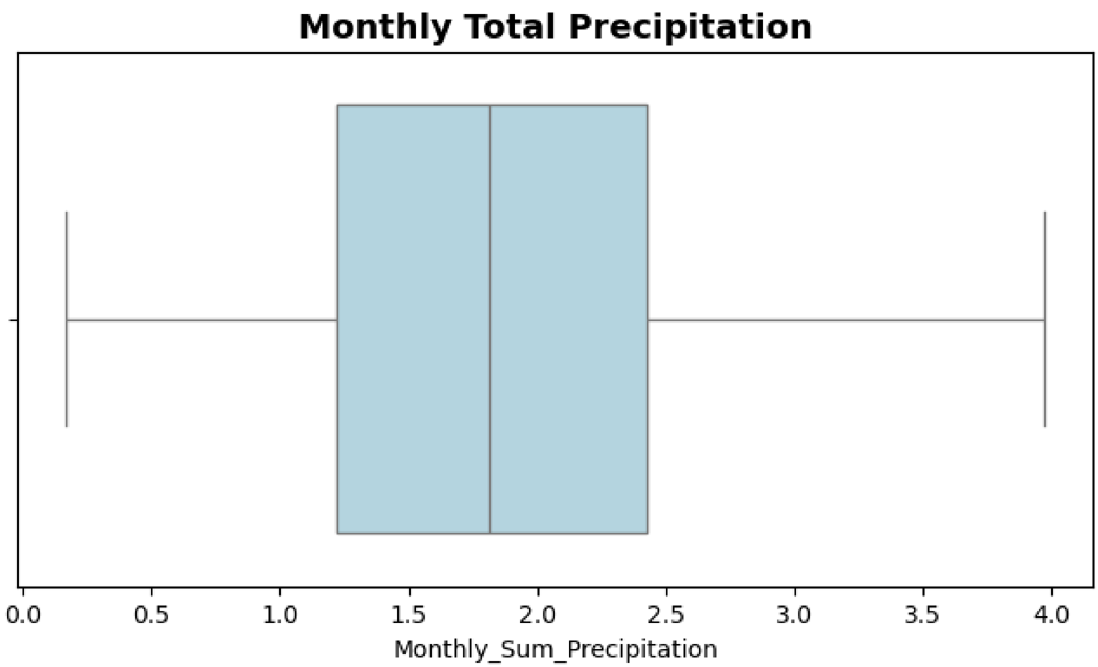
[113]:
```
# Boxplot for Regular Gas Price Average. This distribution is nearly normal␣
↪with a very slight right-skew
plt.figure(figsize=(8, 4))
sns.boxplot(x=df["Regular Gas Price Average"], color="lightblue")
plt.title("Regular Gas Price Average", fontsize=14, fontweight='bold')
plt.show()
```



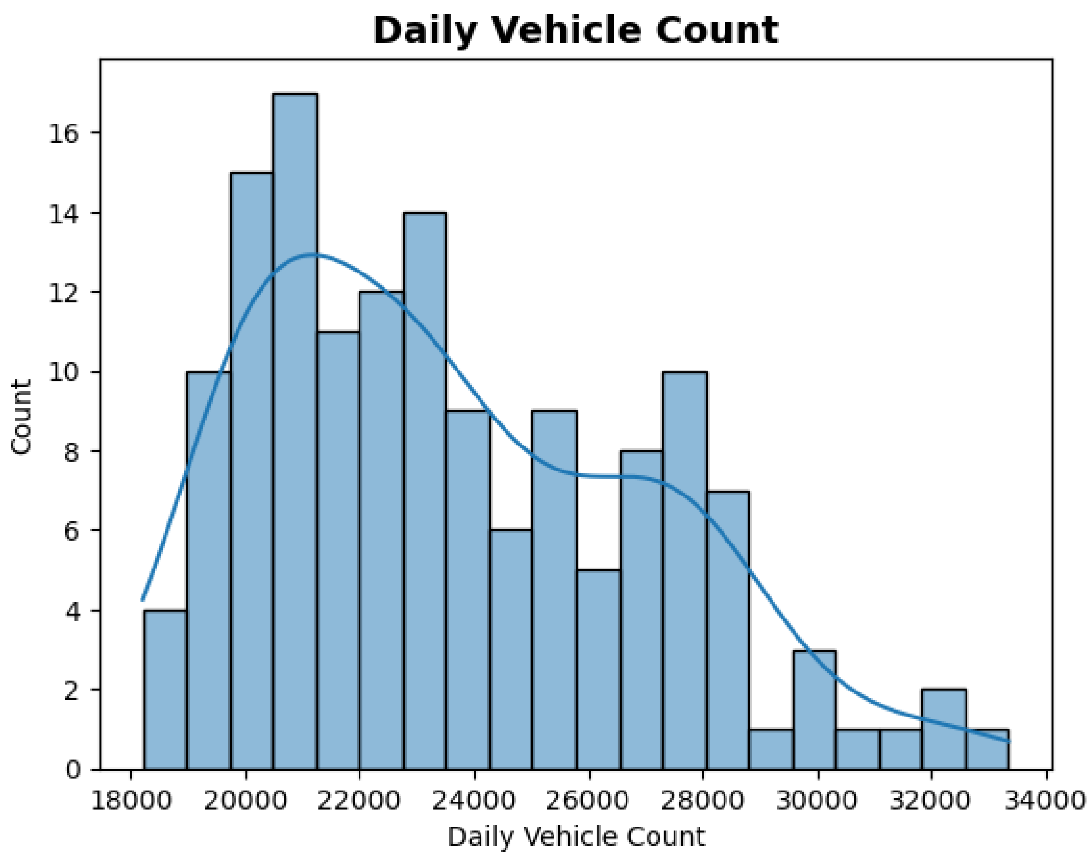
[151]:
```
# Boxplot for Monthly Snowfall. This confirms the previously indication that␣
↪the distribution is very right-skewed
# with some strong outliers
plt.figure(figsize=(8, 4))
sns.boxplot(x=df["Monthly_Snowfall"], color="lightblue")
plt.title("Monthly Snowfall", fontsize=14, fontweight='bold')
plt.show()
```



[244]:
```
# Boxplot for Monthly_Sum_Precipitation. This shows a relatively normal␣
↪distribution of total precipitation amount
plt.figure(figsize=(8, 4))
sns.boxplot(x=df["Monthly_Sum_Precipitation"], color="lightblue")
plt.title("Monthly Total Precipitation", fontsize=14, fontweight='bold')
plt.show()
```



[165]:
```
# Histogram for Daily Vehicle Count which is the dependent variable. Confirms␣
↪again the right-skewed distribution
sns.histplot(data=df, x=df['Daily Vehicle Count'], kde=True, bins=20,␣
↪element="bars")
plt.title("Daily Vehicle Count", fontsize=14, fontweight='bold')
plt.show()
```



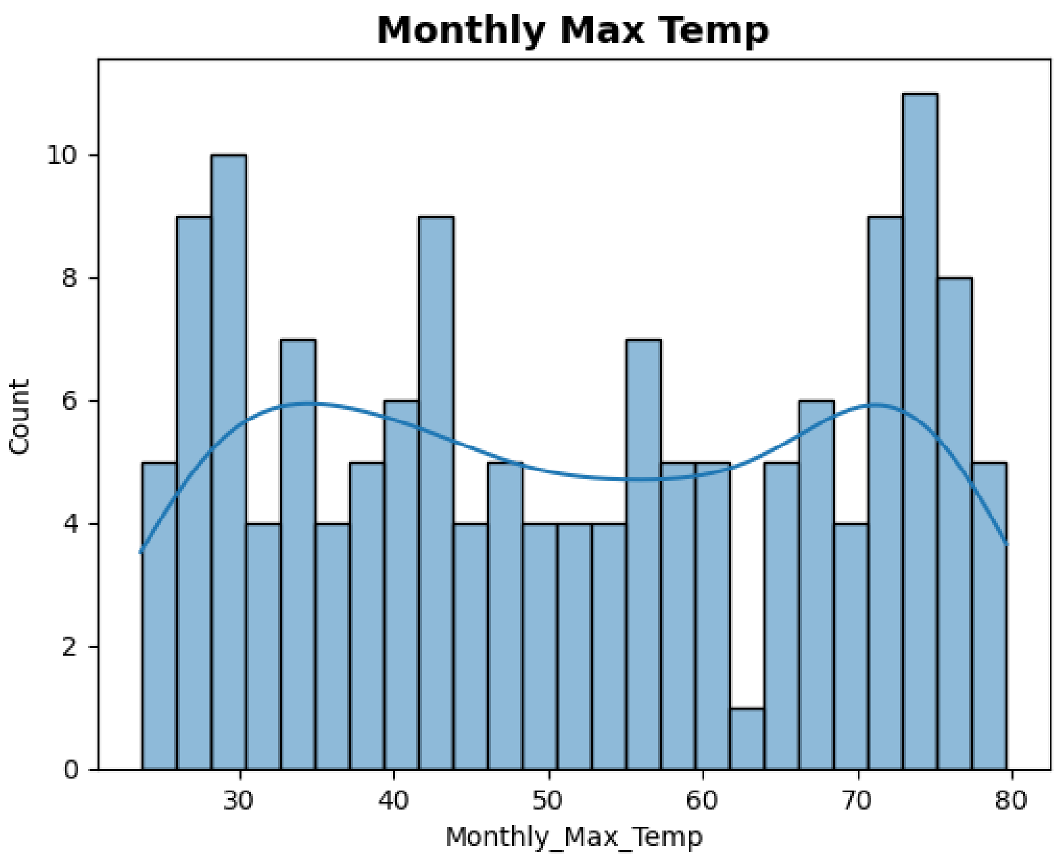
[209]:
```
df["Daily Vehicle Count"].kurt()
```

[209]: -0.37514972255977286

[211]:
```
df["Daily Vehicle Count"].skew()
```

[211]: 0.6271467490447601

[203]:
```
# Histogram for Monthly Max Temp which is an independent variable. This␣
↪variable seems flatly distributed. Will review kurtosis value, which
# should be under 3
sns.histplot(data=df, x=df['Monthly_Max_Temp'], kde=True, bins=25,␣
↪element="bars")
plt.title("Monthly Max Temp", fontsize=14, fontweight='bold')
plt.show()
```



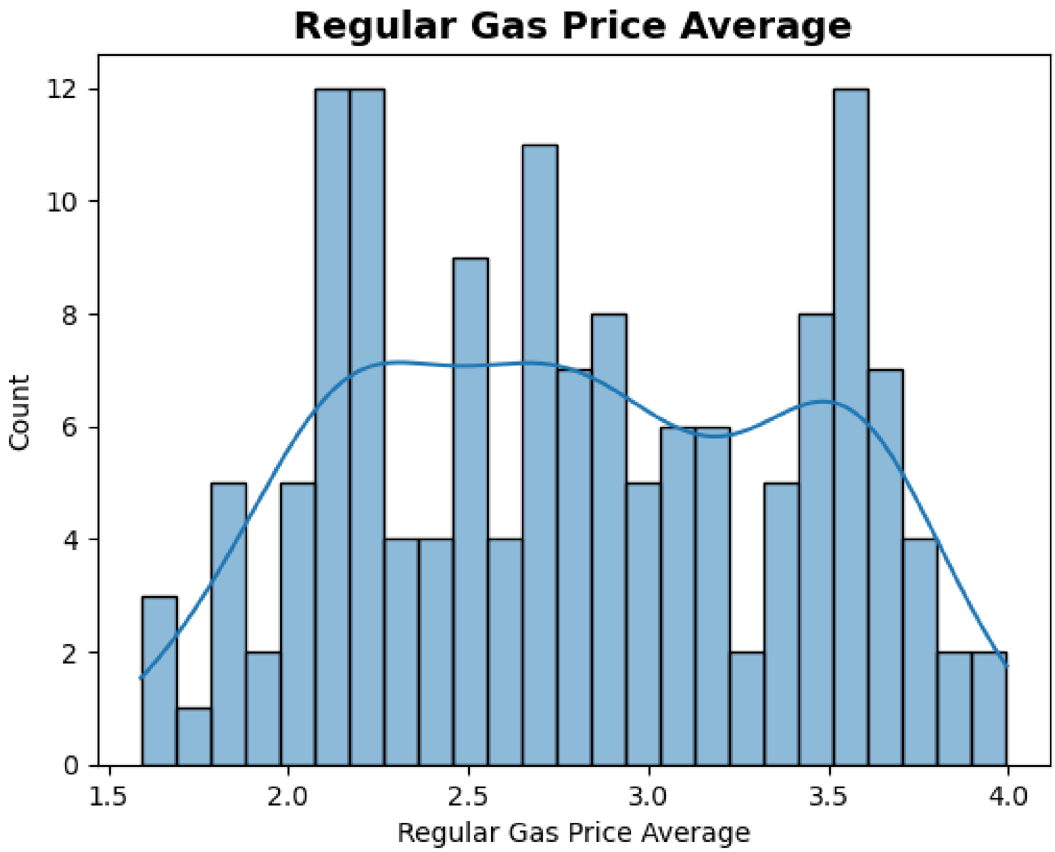
[205]:
```
df["Monthly_Max_Temp"].kurt()
```

[205]: -1.4059379564006056

[207]:
```
df["Monthly_Max_Temp"].skew()
```

[207]: 0.026509500308662615

[189]:
```
# Histogram for Regular Gas Price Average which is an independent variable
sns.histplot(data=df, x=df['Regular Gas Price Average'], kde=True, bins=25,␣
 ↪element="bars")
plt.title("Regular Gas Price Average", fontsize=14, fontweight='bold')
plt.show()
```



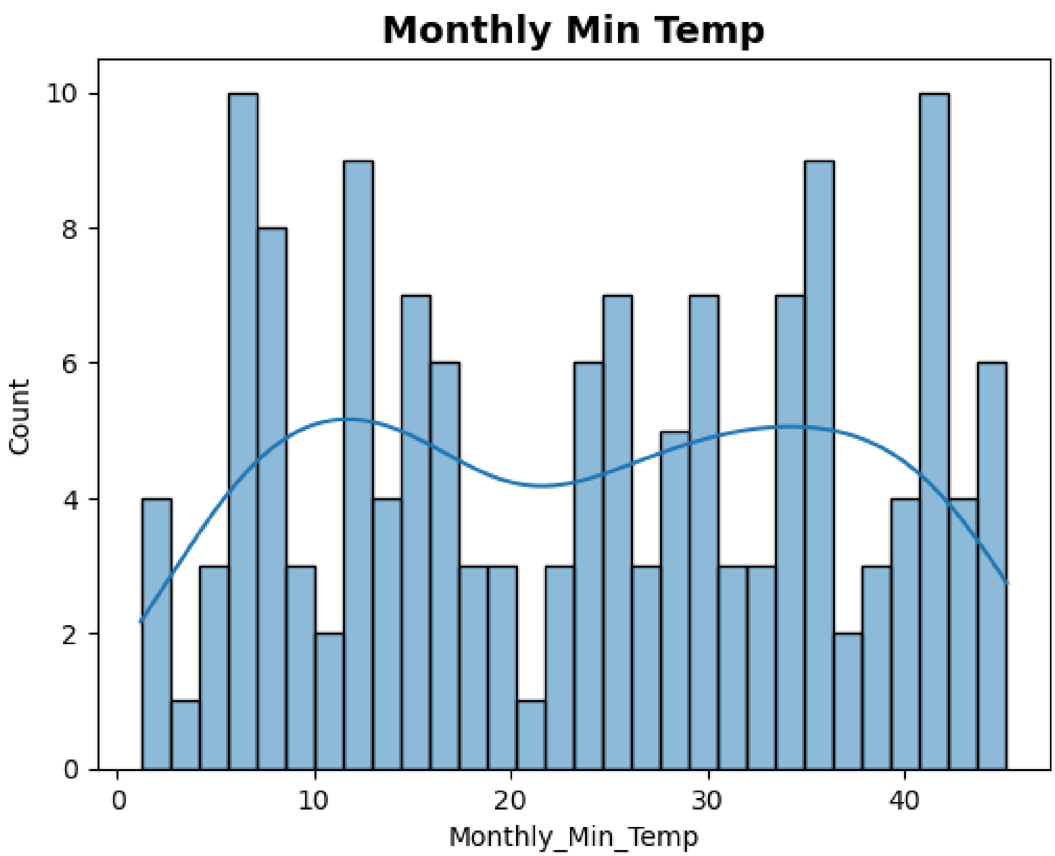
[213]:
```
df["Regular Gas Price Average"].kurt()
```

[213]: -1.1138963533716149

[215]:
```
df["Regular Gas Price Average"].skew()
```

[215]: 0.03347789737537622

[173]:
```
# Histogram for Monthly Minimum Temp which is an independent variable
sns.histplot(data=df, x=df['Monthly_Min_Temp'], kde=True, bins=30,␣
 ↪element="bars")
plt.title("Monthly Min Temp", fontsize=14, fontweight='bold')
plt.show()
```



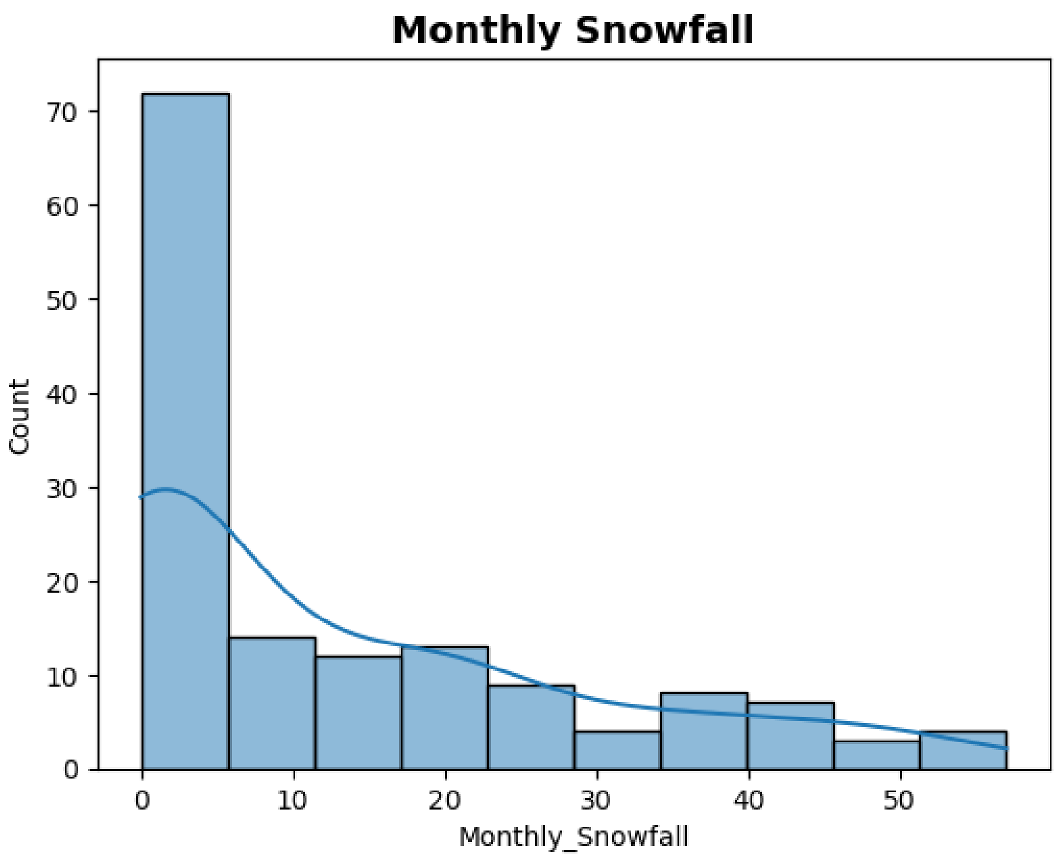
[217]:
```
df["Monthly_Min_Temp"].kurt()
```

[217]: -1.3023763929648215

[219]:
```
df["Monthly_Min_Temp"].skew()
```

[219]: -0.005994346762926201

[153]:
```
# Histogram for Monthly Snowfall which is an independent variable
sns.histplot(data=df, x=df['Monthly_Snowfall'], kde=True, bins=10,␣
 ↪element="bars")
plt.title("Monthly Snowfall", fontsize=14, fontweight='bold')
plt.show()
```



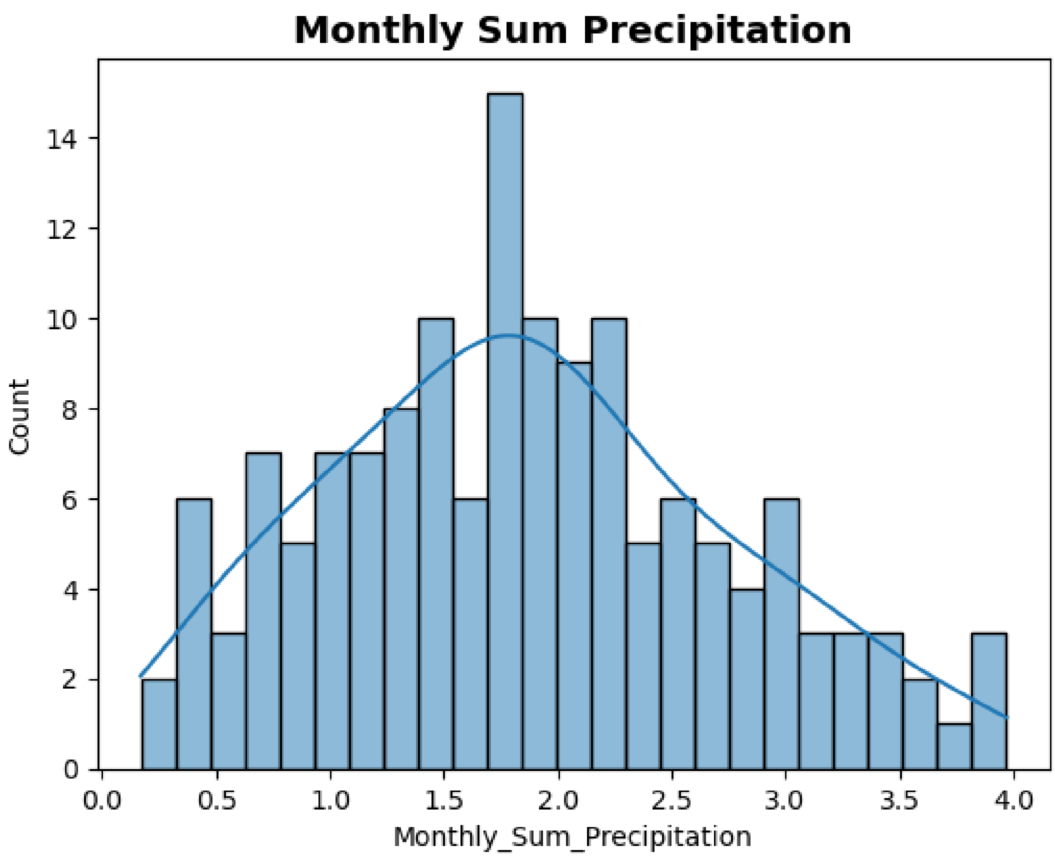
[221]:
```
df["Monthly_Snowfall"].kurt()
```

[221]: 0.1369040985585266

[223]:
```
df["Monthly_Snowfall"].skew()
```

[223]: 1.095048329797924

[179]:
```
# Histogram for Monthly Sum Precipitation which is an independent variable
sns.histplot(data=df, x=df['Monthly_Sum_Precipitation'], kde=True, bins=25,␣
 ↪element="bars")
plt.title("Monthly Sum Precipitation", fontsize=14, fontweight='bold')
plt.show()
```



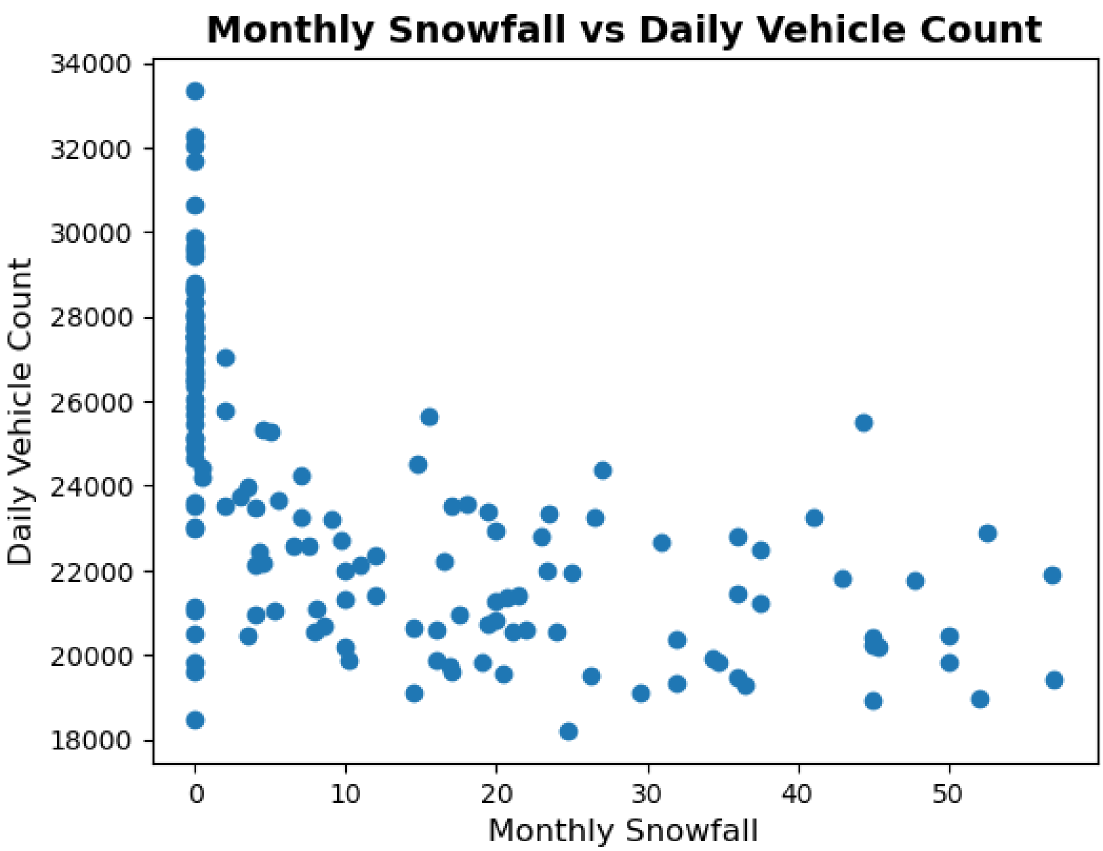
[225]:
```
df["Monthly_Sum_Precipitation"].kurt()
```

[225]: -0.4639077846217101

[227]:
```
df["Monthly_Sum_Precipitation"].skew()
```

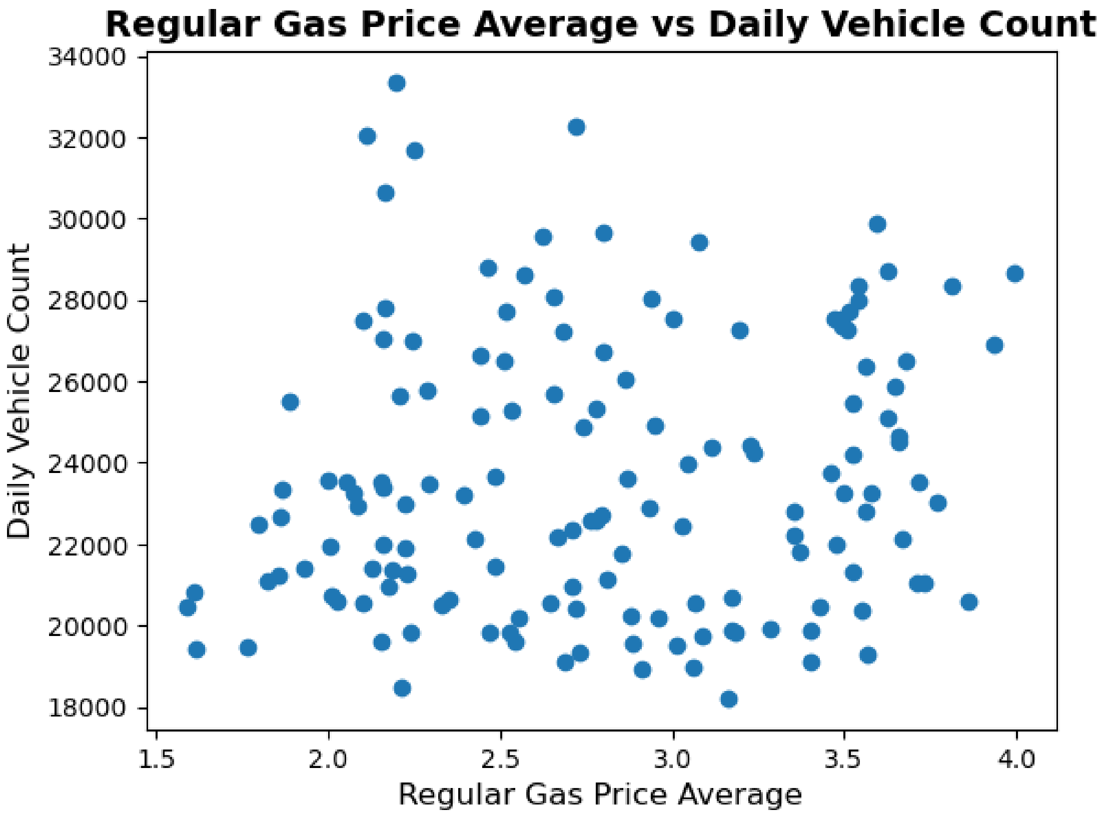
[227]: 0.2806363248683508

[235]:
```
#Scatterplot Monthly Snowfall vs Daily Vehicle Count
plt.scatter(x=df['Monthly_Snowfall'], y=df['Daily Vehicle Count'])
plt.title('Monthly Snowfall vs Daily Vehicle Count', fontsize=14,␣
  ↪fontweight='bold')
plt.xlabel("Monthly Snowfall", fontsize=12)
plt.ylabel("Daily Vehicle Count", fontsize=12)
plt.show()
```

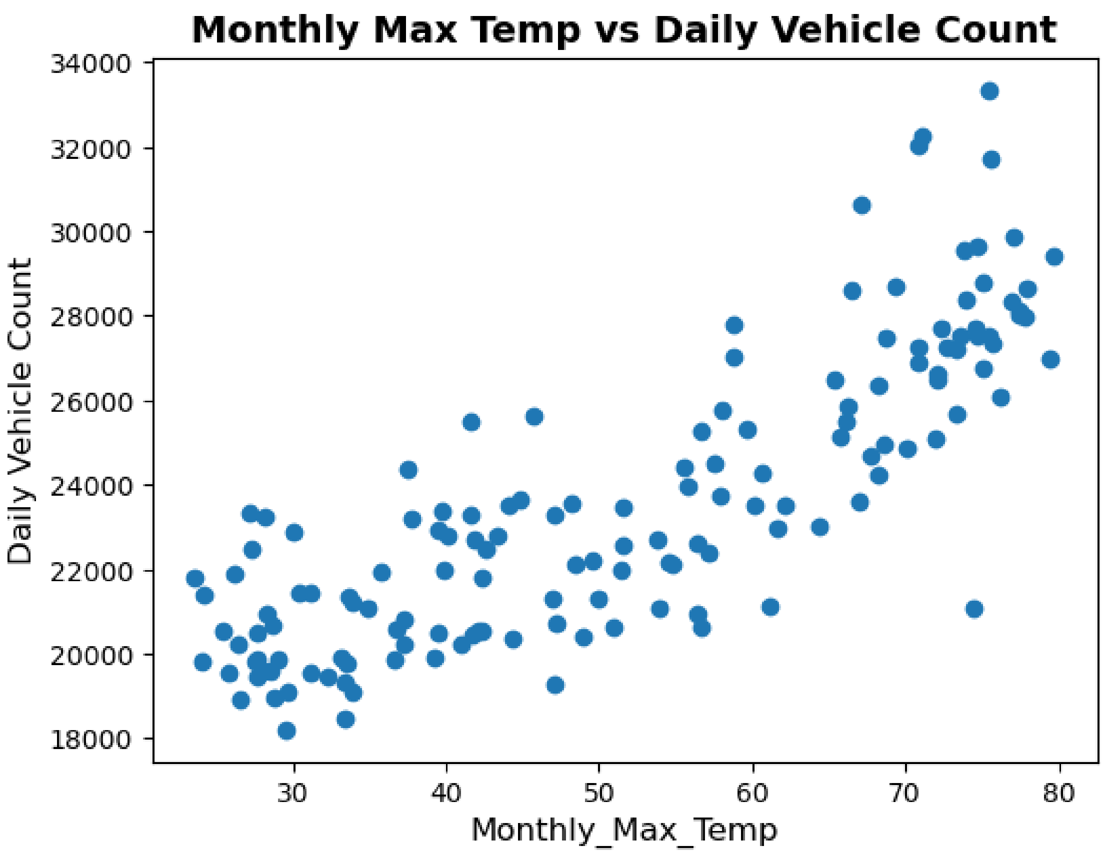


[237]:

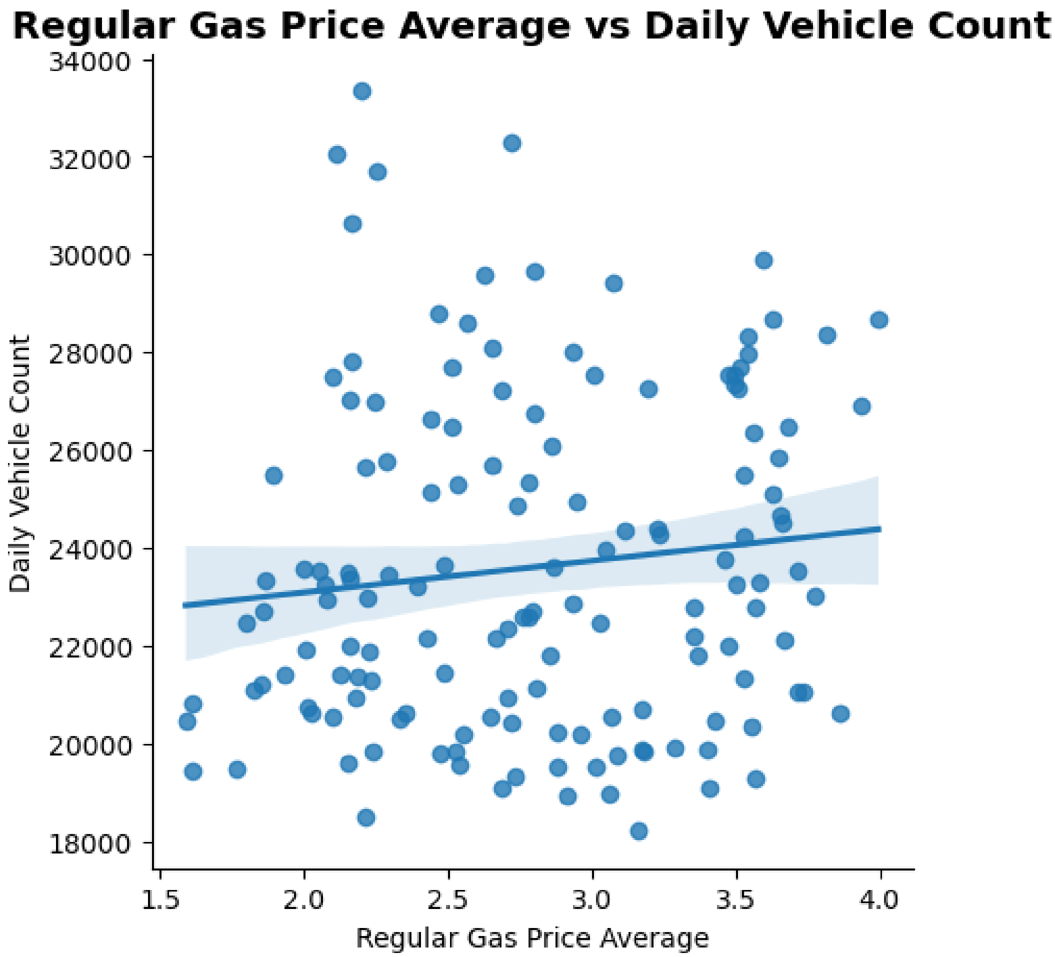
```
#Scatterplot Monthly Snowfall vs Daily Vehicle Count
plt.scatter(x=df['Regular Gas Price Average'], y=df['Daily Vehicle Count'])
plt.title('Regular Gas Price Average vs Daily Vehicle Count', fontsize=14,␣
  ↪fontweight='bold')
plt.xlabel("Regular Gas Price Average", fontsize=12)
plt.ylabel("Daily Vehicle Count", fontsize=12)
plt.show()
```



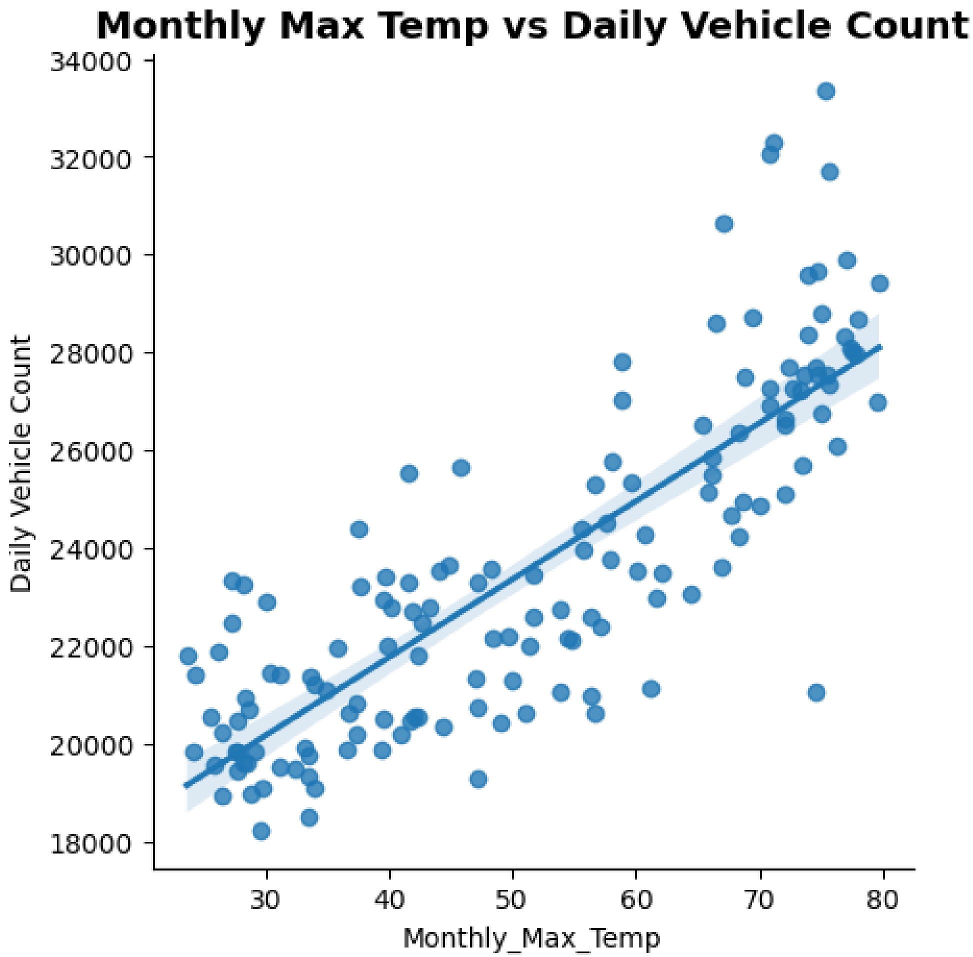
[239]:
```
#Scatterplot Monthly Snowfall vs Daily Vehicle Count
plt.scatter(x=df['Monthly_Max_Temp'], y=df['Daily Vehicle Count'])
plt.title('Monthly Max Temp vs Daily Vehicle Count', fontsize=14,␣
 ↪fontweight='bold')
plt.xlabel("Monthly_Max_Temp", fontsize=12)
plt.ylabel("Daily Vehicle Count", fontsize=12)
plt.show()
```



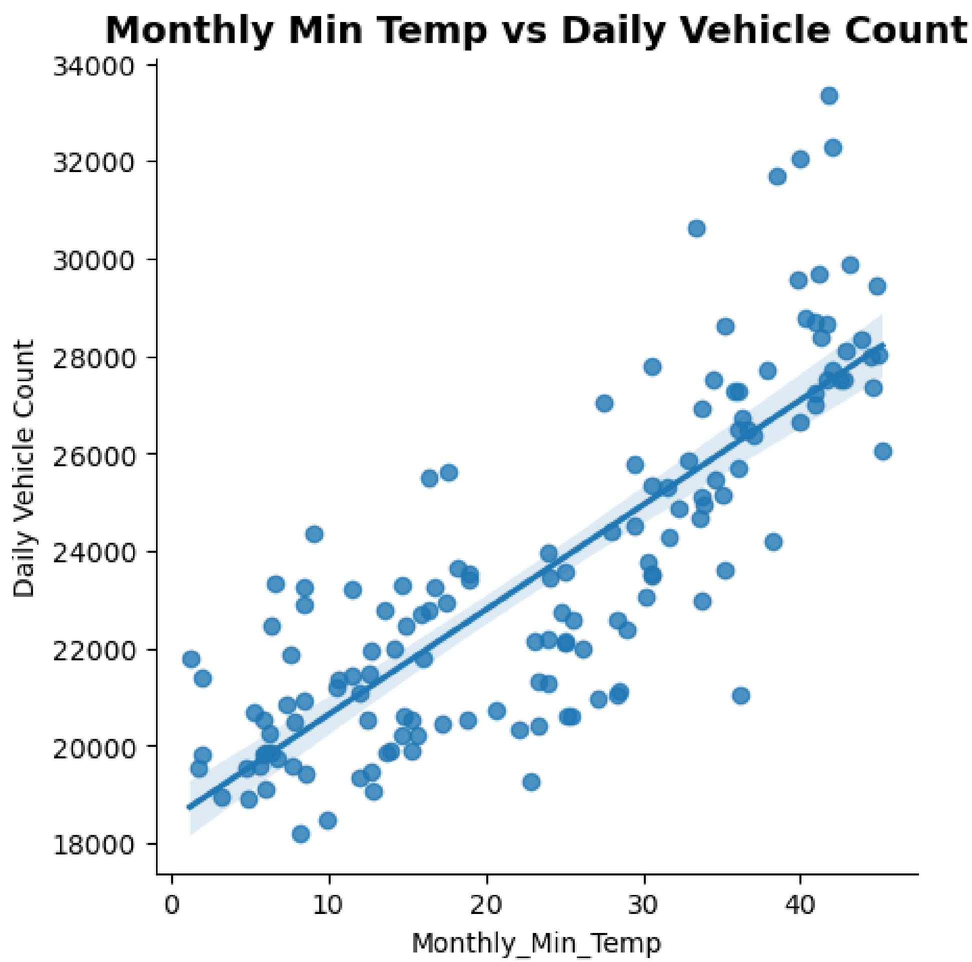
[123]:
```
sns.lmplot(x = 'Regular Gas Price Average',y = "Daily Vehicle Count", data=df)
plt.title("Regular Gas Price Average vs Daily Vehicle Count", fontsize=14,␣
  ↪fontweight='bold')
plt.show()
```



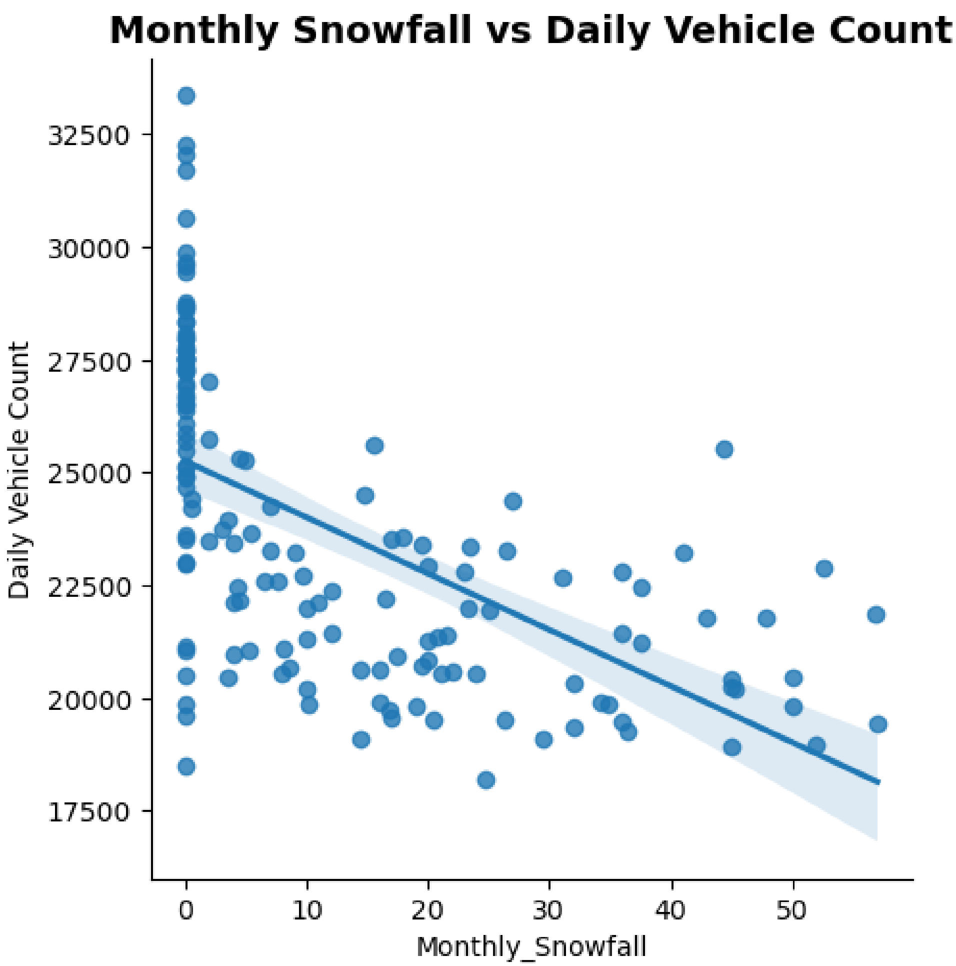
[127]:
```
sns.lmplot(x = 'Monthly_Max_Temp',y = "Daily Vehicle Count", data=df)
plt.title("Monthly Max Temp vs Daily Vehicle Count", fontsize=14,␣
  ↪fontweight='bold')
plt.show()
```



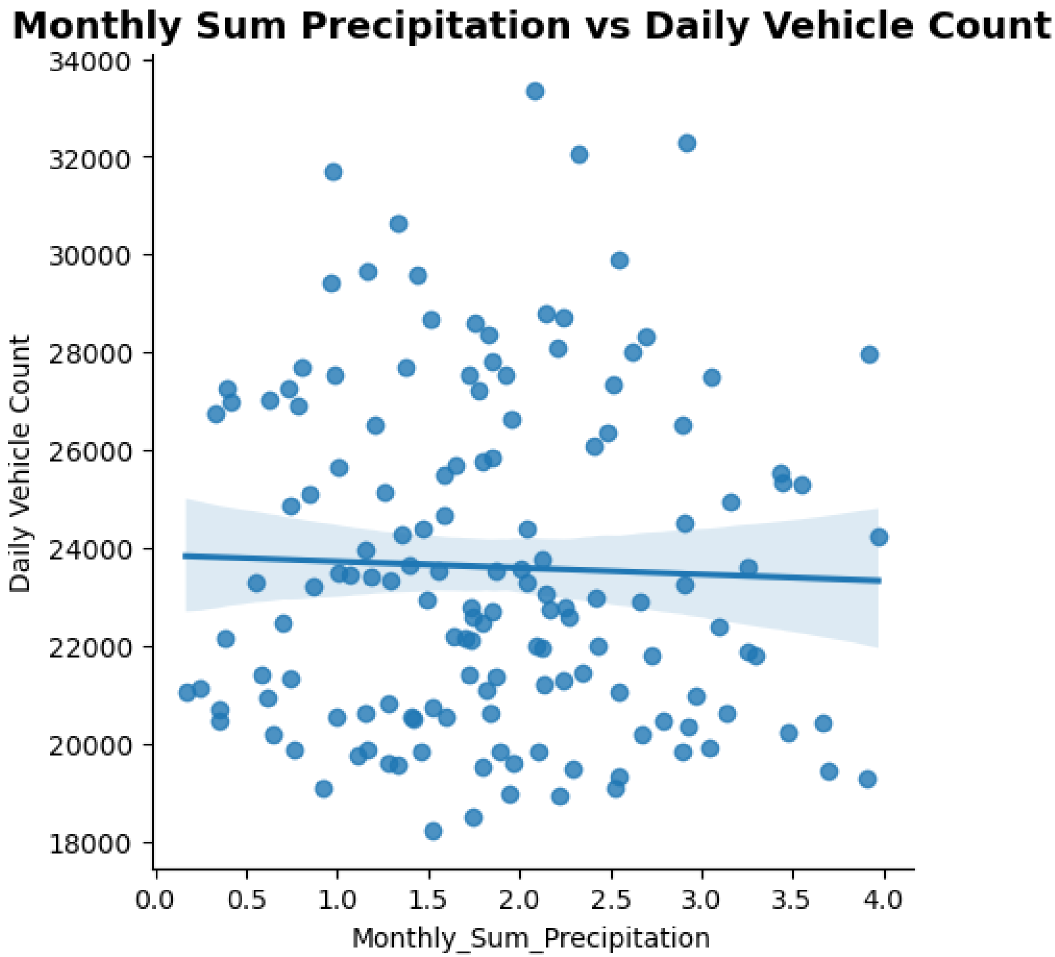
[129]:
```
sns.lmplot(x = 'Monthly_Min_Temp',y = "Daily Vehicle Count", data=df)
plt.title("Monthly Min Temp vs Daily Vehicle Count", fontsize=14,␣
 ↪fontweight='bold')
plt.show()
```



[131]:
```
sns.lmplot(x = 'Monthly_Snowfall',y = "Daily Vehicle Count", data=df)
plt.title("Monthly Snowfall vs Daily Vehicle Count", fontsize=14,␣
 ↪fontweight='bold')
plt.show()
```

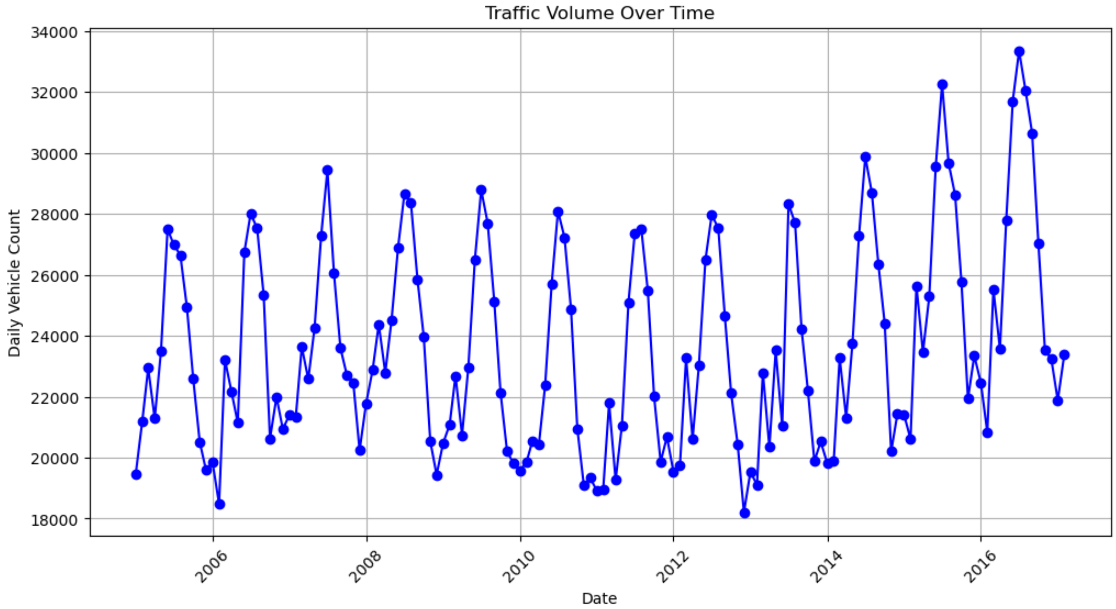


[133]:
```
sns.lmplot(x = 'Monthly_Sum_Precipitation',y = "Daily Vehicle Count", data=df)
plt.title("Monthly Sum Precipitation vs Daily Vehicle Count", fontsize=14,␣
 ↪fontweight='bold')
plt.show()
```

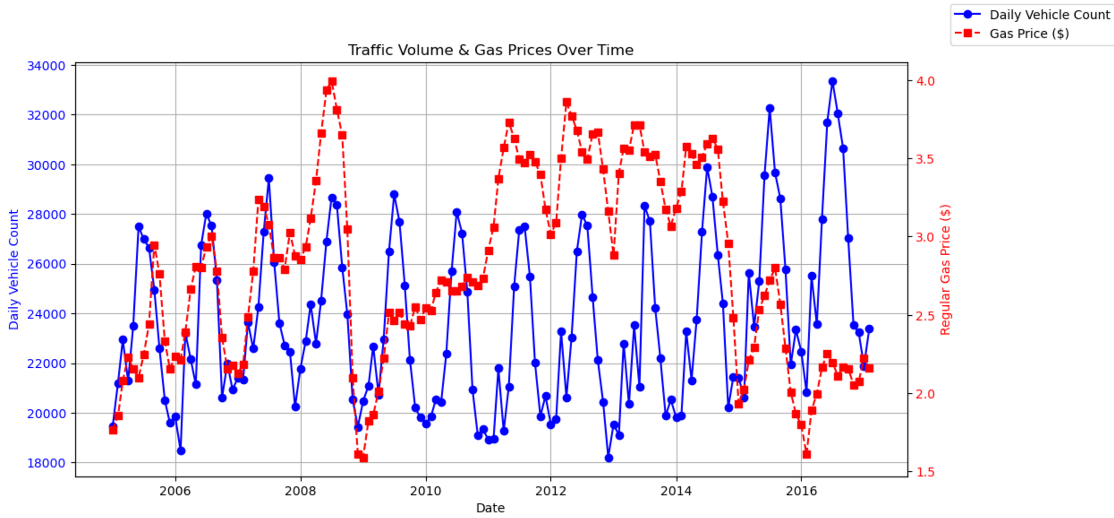


[193]:

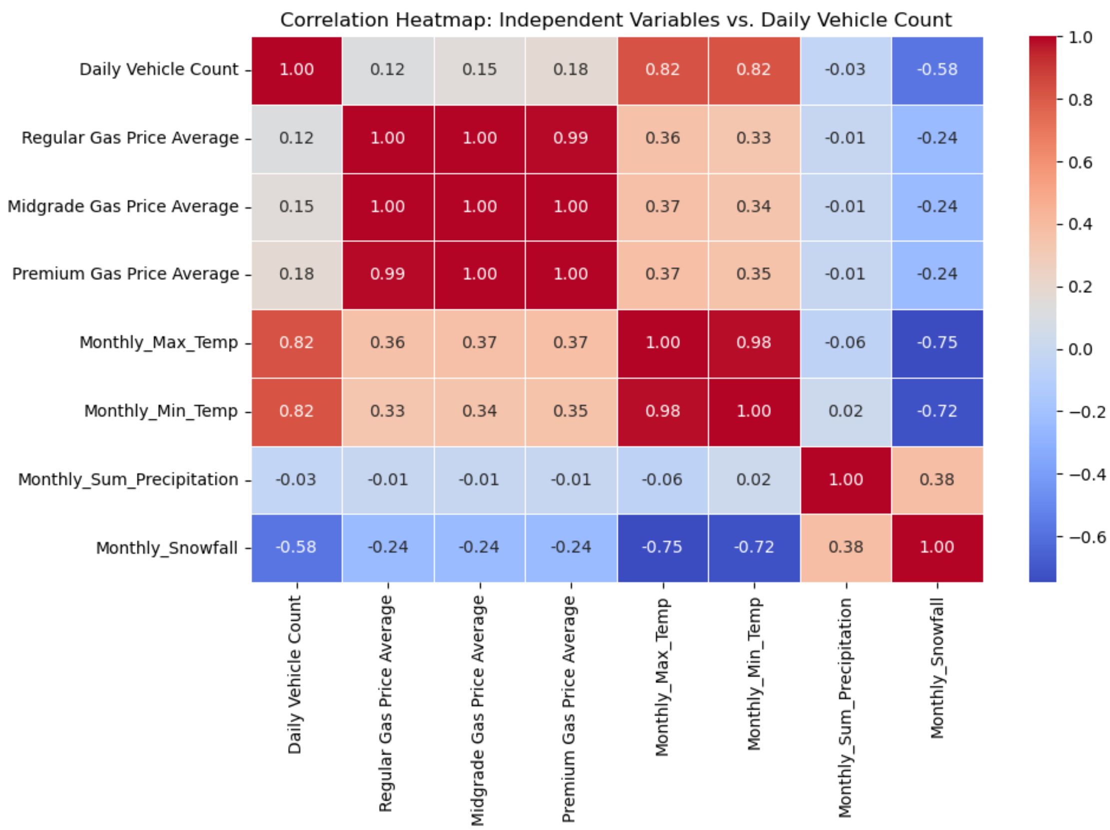
```
# Plot Daily Vehicle Count over time
plt.figure(figsize=(12,6))
plt.plot(df.index, df['Daily Vehicle Count'], marker="o", linestyle="-",␣
 ↪color="blue")

# Formatting
plt.xlabel("Date")
plt.ylabel("Daily Vehicle Count")
plt.title("Traffic Volume Over Time")
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.grid()

plt.show()
```



[197]:
```
# Create figure and axis
fig, ax1 = plt.subplots(figsize=(12,6))

# Plot Daily Vehicle Count on primary y-axis
ax1.plot(df.index, df['Daily Vehicle Count'], marker="o", linestyle="-",␣
 ↪color="blue", label="Daily Vehicle Count")
ax1.set_xlabel("Date")
ax1.set_ylabel("Daily Vehicle Count", color="blue")
ax1.tick_params(axis="y", labelcolor="blue")
ax1.grid()

# Create secondary y-axis for Gas Price
ax2 = ax1.twinx()
ax2.plot(df.index, df['Regular Gas Price Average'], marker="s",␣
 ↪linestyle="dashed", color="red", label="Gas Price ($)")
ax2.set_ylabel("Regular Gas Price ($)", color="red")
ax2.tick_params(axis="y", labelcolor="red")

# Title and legend
plt.title("Traffic Volume & Gas Prices Over Time")
fig.legend(loc="upper right", bbox_to_anchor=(1.1, 1))

plt.show()
```

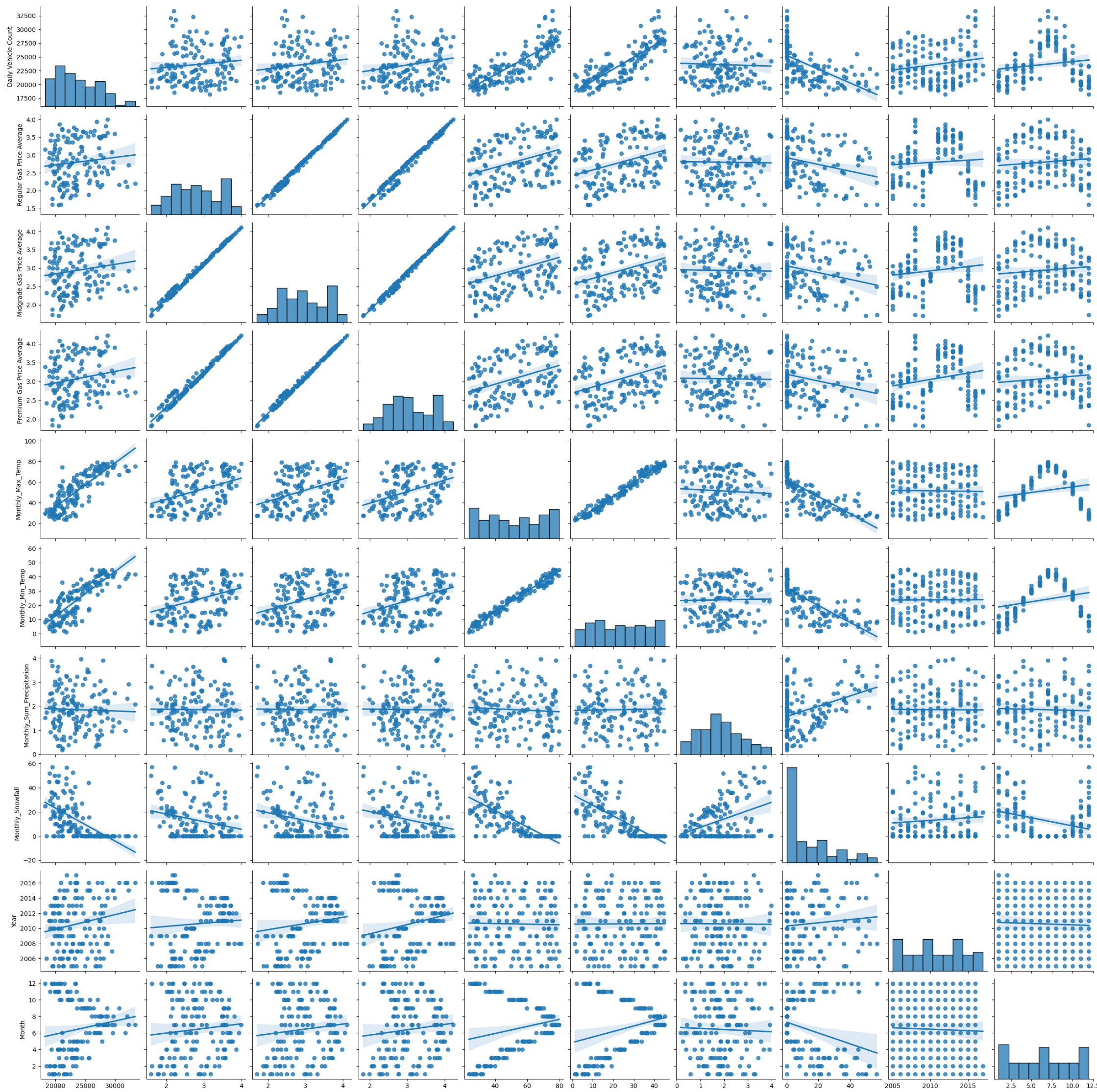


[201]:
```
# Select only numerical independent variables + Daily Vehicle Count
corr_data = df[['Daily Vehicle Count', 'Regular Gas Price Average', 'Midgrade␣
↪Gas Price Average',
                'Premium Gas Price Average', 'Monthly_Max_Temp',␣
↪'Monthly_Min_Temp',
                'Monthly_Sum_Precipitation', 'Monthly_Snowfall']]

# Compute correlation matrix
corr_matrix = corr_data.corr()

# Set up figure size
plt.figure(figsize=(10, 6))

# Create heatmap
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Formatting
plt.title("Correlation Heatmap: Independent Variables vs. Daily Vehicle Count")

plt.show()
```



[242]:
```
sns.pairplot(df, kind="reg")
```

[242]: <seaborn.axisgrid.PairGrid at 0x7f7b98c95300>


Daily Vehicle Count
Regular Gas Price Average
Midgrade Gas Price Average
Premium Gas Price Average
Monthly_Max_Temp
Monthly_Min_Temp
Monthly_Sum_Precipitation
Monthly_Snowfall
Year
Month

Congestion_Simple_Model

March 9, 2025

[1]:
```python
# Importing necessary libraries for time series analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.tsa.api import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statsmodels.api as sm
import seaborn as sns
```

[3]:
```python
# Read in the data
df = pd.read_csv('/Users/helenamabey/Stats_Spring_2025/Congestion Cleaned.csv')
df.head()
```

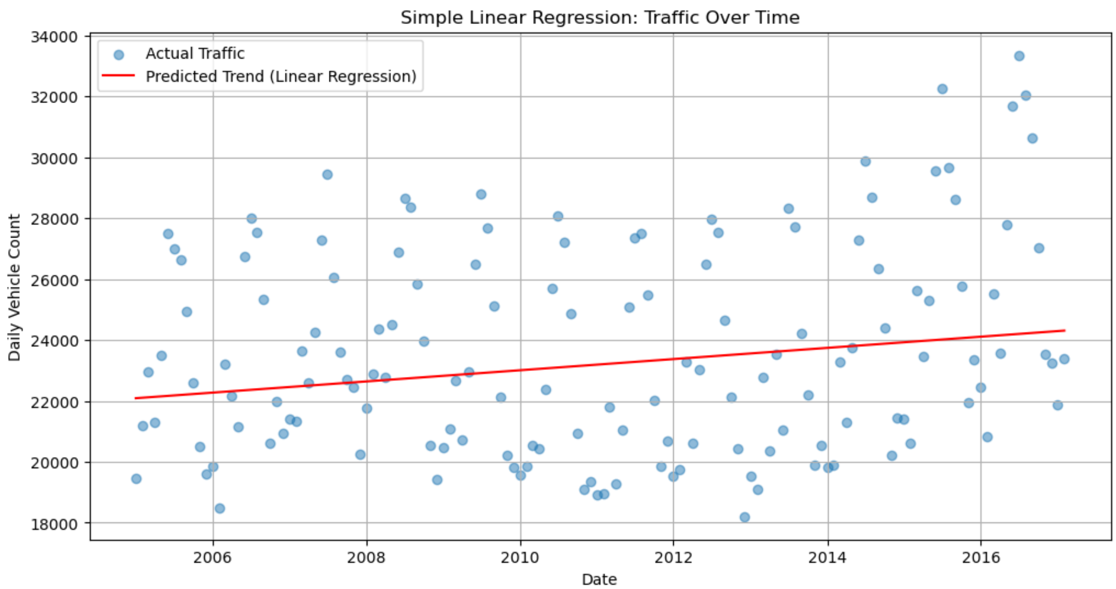
[3]:
```
         Date Month Year  Daily Vehicle Count  Regular Gas Price Average  \
0  2005-01-01    2005-01                19470                     1.7660
1  2005-02-01    2005-02                21207                     1.8550
2  2005-03-01    2005-03                22943                     2.0825
3  2005-04-01    2005-04                21288                     2.2300
4  2005-05-01    2005-05                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[5]:
```python
# Convert 'Date' to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

[7]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Ensure 'Date' is in datetime format
df.index = pd.to_datetime(df.index)

# Convert Date to a numeric value (days since first date)
df['Days_Since_Start'] = (df.index - df.index.min()).days

# Define independent (X) and dependent (y) variables
X = df[['Days_Since_Start']]  # Independent variable (time)
y = df['Daily Vehicle Count']  # Dependent variable (traffic volume)

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)

# Fit the simple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model performance
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")

# Get predictions for full dataset to visualize trend
df['Predicted_Traffic'] = model.predict(X)

# Plot actual vs. predicted traffic over time
plt.figure(figsize=(12, 6))
plt.scatter(df.index, df['Daily Vehicle Count'], label="Actual Traffic",␣
↪alpha=0.5)
plt.plot(df.index, df['Predicted_Traffic'], label="Predicted Trend (Linear␣
↪Regression)", color="red")
plt.xlabel("Date")
plt.ylabel("Daily Vehicle Count")
```

```python
plt.title("Simple Linear Regression: Traffic Over Time")
plt.legend()
plt.grid()
plt.show()
```

```
R² Score: -0.2148
Mean Squared Error: 17612282.17
```



[9]:
```python
import statsmodels.api as sm

# Add a constant for intercept
X_with_const = sm.add_constant(X)

# Fit OLS regression model
model_sm = sm.OLS(y, X_with_const).fit()

# Print model summary
print(model_sm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.039
Model:                             OLS   Adj. R-squared:                  0.032
Method:                  Least Squares   F-statistic:                     5.823
Date:                 Sun, 09 Mar 2025   Prob (F-statistic):             0.0171
Time:                         09:26:58   Log-Likelihood:                -1390.7
No. Observations:                  146   AIC:                             2785.
Df Residuals:                      144   BIC:                             2791.
```

```
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const             2.246e+04    549.428     40.877      0.000    2.14e+04
2.35e+04
Days_Since_Start     0.5196      0.215      2.413      0.017       0.094
0.945
==============================================================================
Omnibus:                       12.769   Durbin-Watson:                   0.510
Prob(Omnibus):                  0.002   Jarque-Bera (JB):                9.037
Skew:                           0.485   Prob(JB):                       0.0109
Kurtosis:                       2.263   Cond. No.                     5.08e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.08e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:

Monthly_Max_Temp_Model

March 9, 2025

[1]:
```python
# Importing necessary libraries for time series analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.tsa.api import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statsmodels.api as sm
```

[3]:
```python
# Read in the data
df = pd.read_csv('/Users/helenamabey/Stats_Spring_2025/Congestion Cleaned.csv')
df.head()
```

[3]:
```
         Date Month Year  Daily Vehicle Count  Regular Gas Price Average  \
0  2005-01-01    2005-01                19470                     1.7660
1  2005-02-01    2005-02                21207                     1.8550
2  2005-03-01    2005-03                22943                     2.0825
3  2005-04-01    2005-04                21288                     2.2300
4  2005-05-01    2005-05                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[5]:
```
# Convert 'Date' to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

[7]:
```
# Define independent variables (ONLY the selected ones)
X = df[['Monthly_Max_Temp', 'Monthly_Min_Temp', 'Monthly_Sum_Precipitation']]

# Define dependent variable
y = df['Daily Vehicle Count']

# Add a constant for the intercept
X = sm.add_constant(X)
```

[9]:
```
# Train-test split (Train: before 2017, Test: 2017)
train_mask = df.index.year < 2017
X_train, X_test = X[train_mask], X[~train_mask]
y_train, y_test = y[train_mask], y[~train_mask]

# Ensure index alignment before fitting
X_train, y_train = X_train.align(y_train, join='inner', axis=0)
```

[11]:
```
# Fit the OLS model
model_sm = sm.OLS(y_train, X_train).fit()
print(model_sm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.684
Model:                             OLS   Adj. R-squared:                  0.677
Method:                  Least Squares   F-statistic:                     101.1
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           7.33e-35
Time:                         19:29:32   Log-Likelihood:                -1292.4
No. Observations:                  144   AIC:                             2593.
Df Residuals:                      140   BIC:                             2605.
Df Model:                            3
Covariance Type:             nonrobust
================================================================================
=============
                                coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-------------
const                      1.633e+04   1460.966     11.176      0.000
1.34e+04    1.92e+04
Monthly_Max_Temp            108.8485     60.126      1.810      0.072
-10.024     227.721
Monthly_Min_Temp             71.9538     81.097      0.887      0.376
```

```
-88.379     232.287
Monthly_Sum_Precipitation   -39.3121    209.418     -0.188      0.851
-453.342     374.718
==============================================================================
Omnibus:                        7.725   Durbin-Watson:                   0.963
Prob(Omnibus):                  0.021   Jarque-Bera (JB):                8.350
Skew:                           0.404   Prob(JB):                       0.0154
Kurtosis:                       3.859   Cond. No.                         554.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[13]:

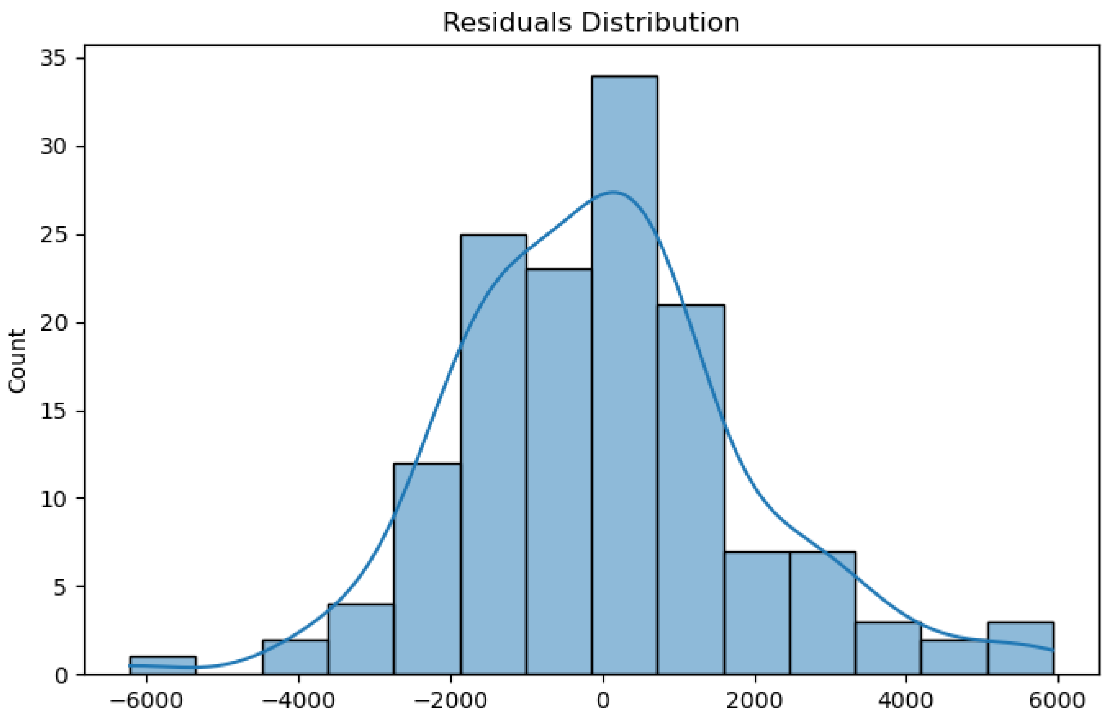
```python
# Keep only the most relevant variable
X_selected = df[['Monthly_Max_Temp']]  # Dropping weak predictors

# Add a constant for the intercept
X_selected = sm.add_constant(X_selected)

# Train-test split (Train: before 2017, Test: 2017)
train_mask = df.index.year < 2017
X_train, X_test = X_selected[train_mask], X_selected[~train_mask]
y_train, y_test = y[train_mask], y[~train_mask]

# Fit the new model
model_sm = sm.OLS(y_train, X_train).fit()
print(model_sm.summary())
```

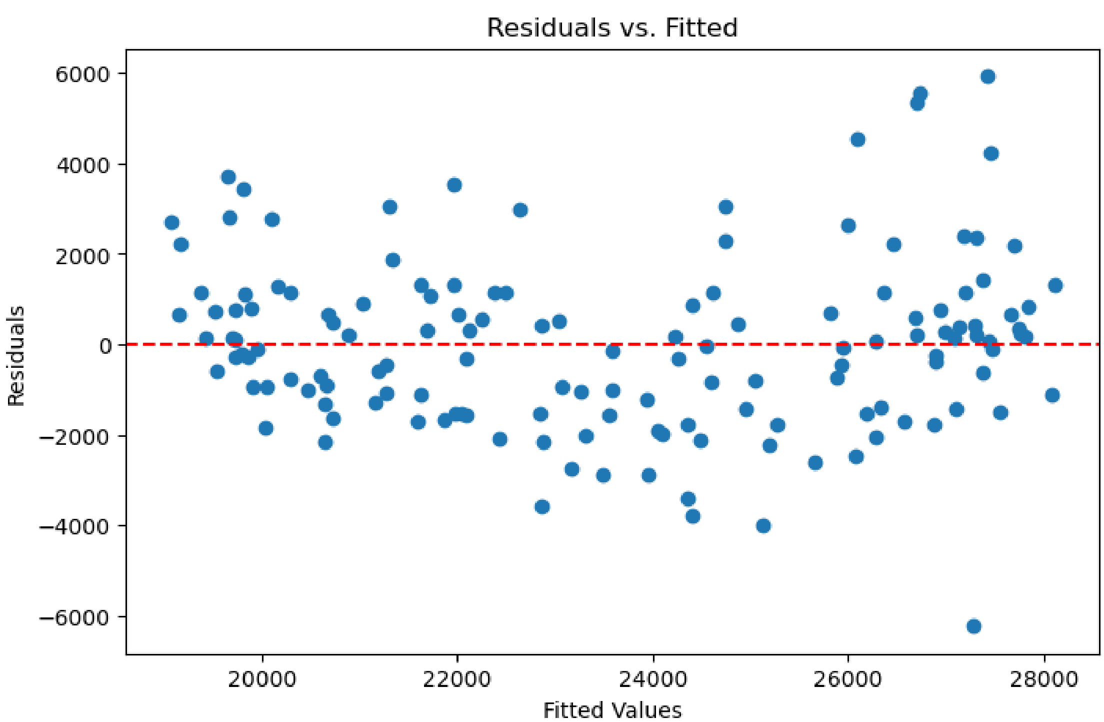
```
                            OLS Regression Results
===============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.682
Model:                             OLS   Adj. R-squared:                  0.680
Method:                  Least Squares   F-statistic:                     304.8
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           3.65e-37
Time:                         19:31:13   Log-Likelihood:                -1292.8
No. Observations:                  144   AIC:                             2590.
Df Residuals:                      142   BIC:                             2596.
Df Model:                            1
Covariance Type:             nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const             1.525e+04    505.540     30.169      0.000    1.43e+04
1.63e+04
```

```
Monthly_Max_Temp   161.4327      9.247     17.458      0.000     143.153
179.712
==============================================================================
Omnibus:                        8.204   Durbin-Watson:                   0.943
Prob(Omnibus):                  0.017   Jarque-Bera (JB):                9.586
Skew:                           0.388   Prob(JB):                      0.00829
Kurtosis:                       3.997   Cond. No.                         172.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[15]:

```python
import seaborn as sns

# Get residuals
residuals = model_sm.resid

# Histogram of residuals
plt.figure(figsize=(8,5))
sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.show()

# Residuals vs. Fitted Values
plt.figure(figsize=(8,5))
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(y=0, color='r', linestyle='dashed')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted")
plt.show()
```


Residuals Distribution
Count
35
30
25
20
15
10
5
0
−6000
−4000
−2000
0
2000
4000
6000

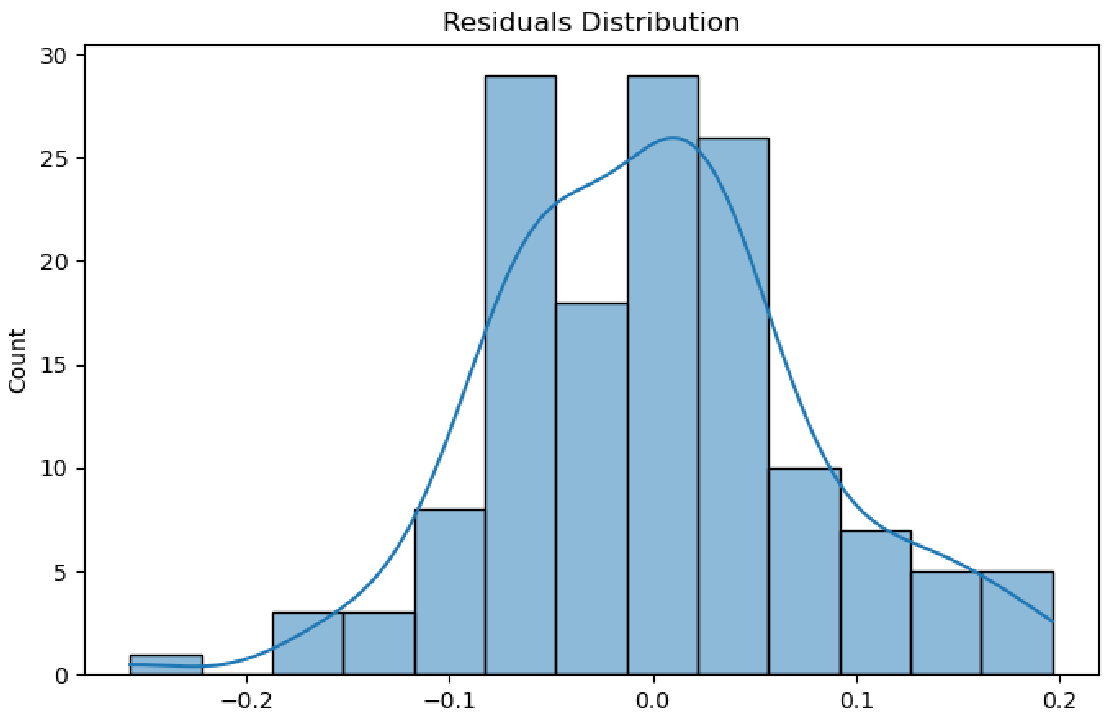

Residuals vs. Fitted
Residuals
6000
4000
2000
0
−2000
−4000
−6000
20000
22000
24000
26000
28000
Fitted Values

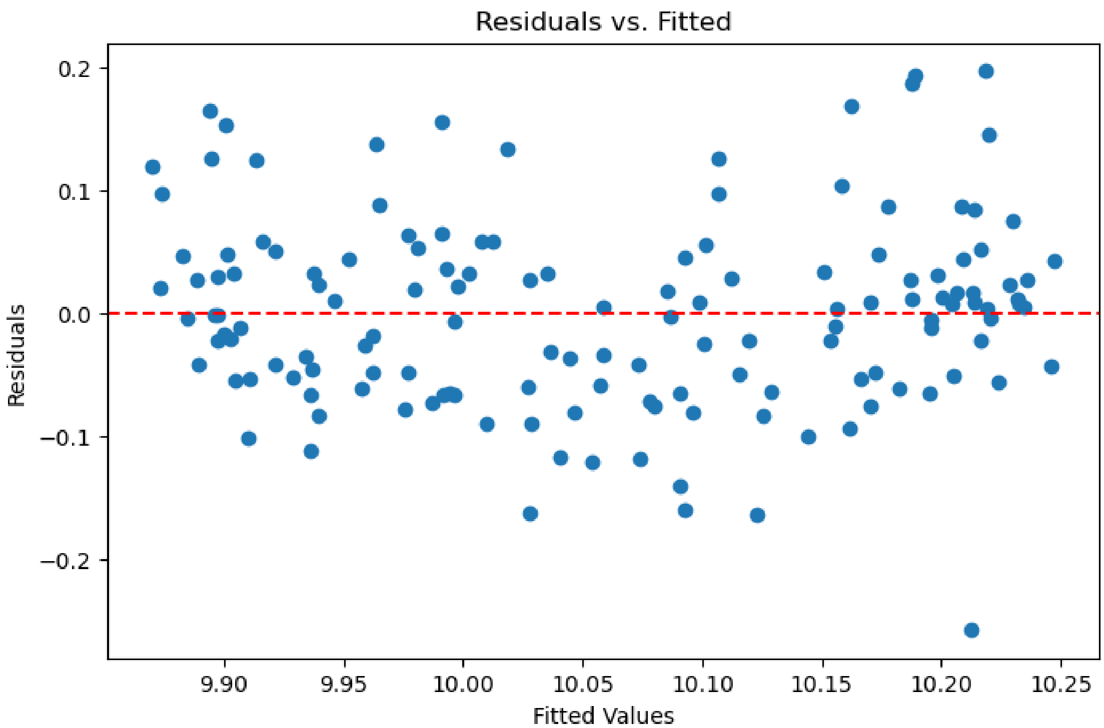
[17]:
```
# Log-transform the target variable
y_train_log = np.log(y_train)

# Fit model with log-transformed target
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.695
Model:                             OLS   Adj. R-squared:                  0.693
Method:                  Least Squares   F-statistic:                     323.6
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           1.94e-38
Time:                         19:34:44   Log-Likelihood:                 163.94
No. Observations:                  144   AIC:                            -323.9
Df Residuals:                      142   BIC:                            -317.9
Df Model:                            1
Covariance Type:             nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const                9.7114      0.020    475.296      0.000       9.671
9.752
Monthly_Max_Temp     0.0067      0.000     17.988      0.000       0.006
0.007
==============================================================================
Omnibus:                        2.277   Durbin-Watson:                   1.029
Prob(Omnibus):                  0.320   Jarque-Bera (JB):                1.883
Skew:                           0.149   Prob(JB):                        0.390
Kurtosis:                       3.474   Cond. No.                         172.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[19]:
```
# Get residuals
residuals_log = model_sm_log.resid

# Histogram of residuals
plt.figure(figsize=(8,5))
sns.histplot(residuals_log, kde=True)
```

```
plt.title("Residuals Distribution")
plt.show()

# Residuals vs. Fitted Values
plt.figure(figsize=(8,5))
plt.scatter(model_sm_log.fittedvalues, residuals_log)
plt.axhline(y=0, color='r', linestyle='dashed')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted")
plt.show()
```





[21]:
```
y_pred_log = model_sm_log.predict(X_test)

# Convert back from log-scale
y_pred_original_scale = np.exp(y_pred_log)

# Compare actual vs predicted values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted':␣
 ↪y_pred_original_scale})
print(predictions_df)
```

```
            Actual     Predicted
Date
2017-01-01   21883  19682.691555
2017-02-01   23391  21552.565382
```

[23]:
```
# Calculate correction factor for log transformation bias
residuals_log = model_sm_log.resid
correction_factor = np.exp(residuals_log.var() / 2)

# Adjust predictions
y_pred_corrected = y_pred_original_scale * correction_factor

# Compare corrected predictions
```

```
predictions_df['Corrected_Predicted'] = y_pred_corrected
print(predictions_df)
```

```
            Actual     Predicted  Corrected_Predicted
Date
2017-01-01   21883  19682.691555         19742.309249
2017-02-01   23391  21552.565382         21617.846812
```

[25]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

# Train Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)

# Compare R² scores
print(f"Linear Model R²: {model_sm_log.rsquared:.4f}")
print(f"Random Forest R²: {r2_score(y_test, y_pred_rf):.4f}")
```

```
Linear Model R²: 0.6950
Random Forest R²: -4.8087
```

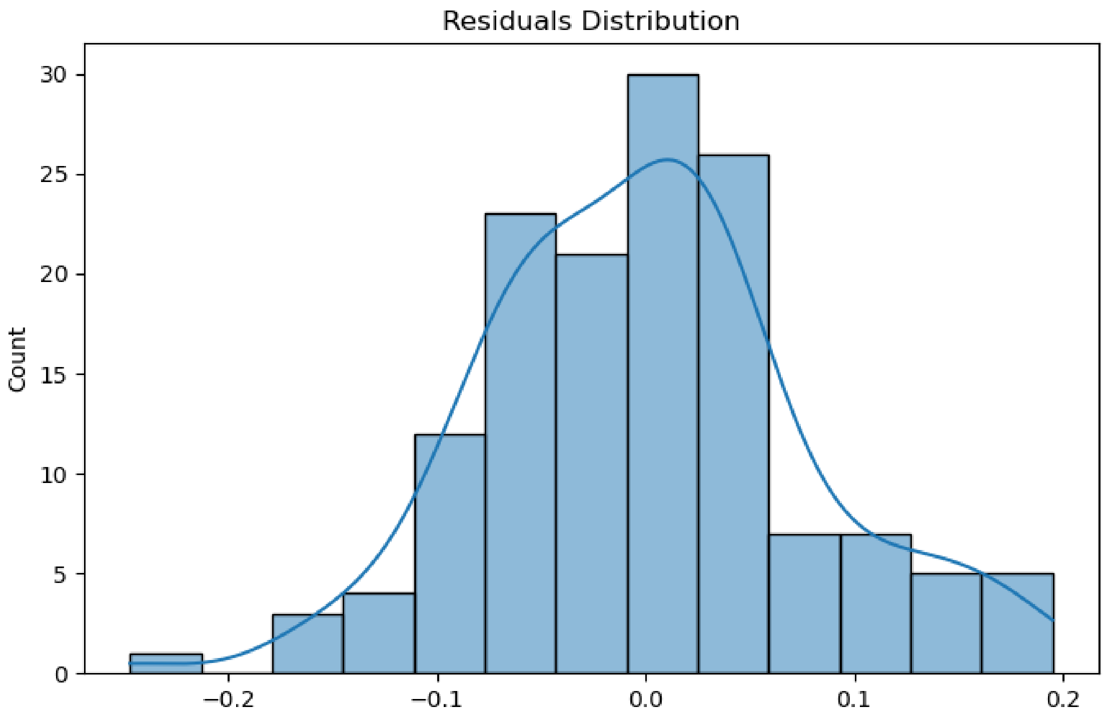
[27]:
```
# Select features (adding back Monthly_Min_Temp)
X_selected = df[['Monthly_Max_Temp', 'Monthly_Min_Temp']]

# Add intercept
X_selected = sm.add_constant(X_selected)

# Train-test split (Train: before 2017, Test: 2017)
train_mask = df.index.year < 2017
X_train, X_test = X_selected[train_mask], X_selected[~train_mask]
y_train, y_test = y[train_mask], y[~train_mask]

# Log-transform the target variable
y_train_log = np.log(y_train)

# Fit model with log-transformed target
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print new model summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.696
```

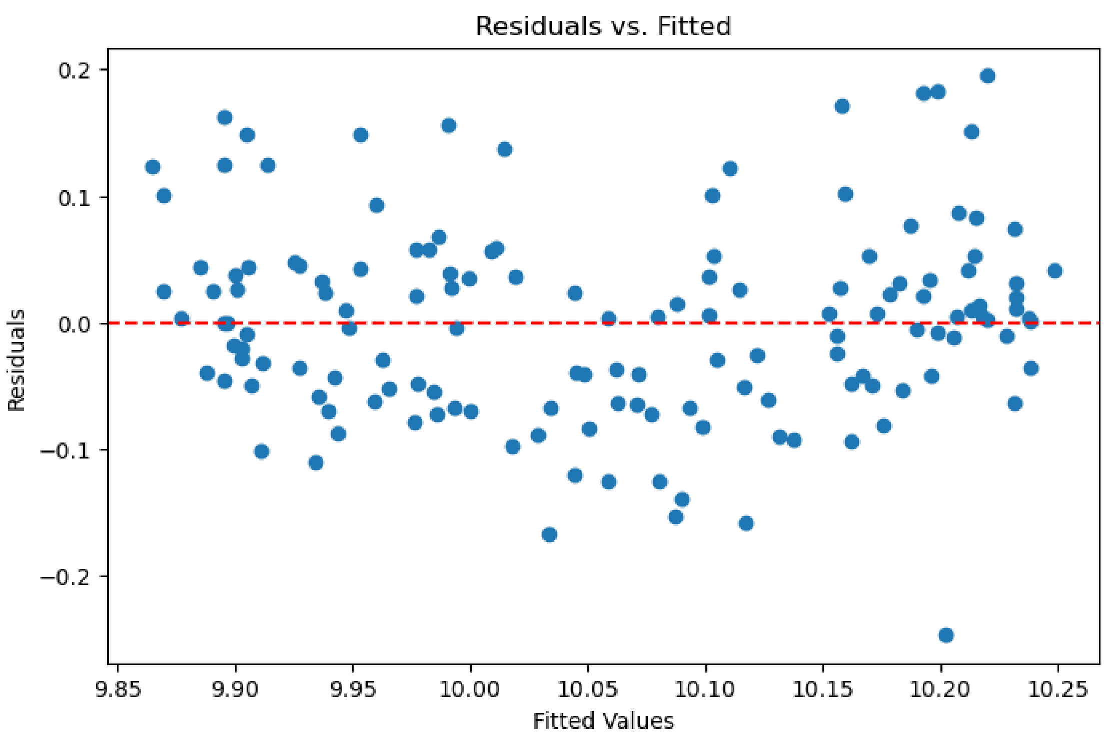
```
Model:                            OLS   Adj. R-squared:                  0.692
Method:                 Least Squares   F-statistic:                     161.7
Date:                Fri, 07 Mar 2025   Prob (F-statistic):           3.19e-37
Time:                        19:41:19   Log-Likelihood:                 164.28
No. Observations:                 144   AIC:                            -322.6
Df Residuals:                     141   BIC:                            -313.6
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const                9.7436      0.045    216.845      0.000       9.655
9.832
Monthly_Max_Temp     0.0050      0.002      2.350      0.020       0.001
0.009
Monthly_Min_Temp     0.0023      0.003      0.806      0.421      -0.003
0.008
==============================================================================
Omnibus:                        1.888   Durbin-Watson:                   1.046
Prob(Omnibus):                  0.389   Jarque-Bera (JB):                1.430
Skew:                           0.163   Prob(JB):                        0.489
Kurtosis:                       3.363   Cond. No.                         421.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[29]:
```python
# Get residuals
residuals_log_2 = model_sm_log.resid

# Histogram of residuals
plt.figure(figsize=(8,5))
sns.histplot(residuals_log_2, kde=True)
plt.title("Residuals Distribution")
plt.show()

# Residuals vs. Fitted Values
plt.figure(figsize=(8,5))
plt.scatter(model_sm_log.fittedvalues, residuals_log_2)
plt.axhline(y=0, color='r', linestyle='dashed')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted")
```

```
plt.show()
```



Residuals Distribution
Count
30
25
20
15
10
5
0
−0.2
−0.1
0.0
0.1
0.2



Residuals vs. Fitted
Residuals
0.2
0.1
0.0
−0.1
−0.2
9.85
9.90
9.95
10.00
10.05
10.10
10.15
10.20
10.25
Fitted Values

[33]:
```
# Predict on test data (log scale)
y_pred_log = model_sm_log.predict(X_test)

# Convert back to original scale (with correction factor)
correction_factor = np.exp(model_sm_log.resid.var() / 2)
y_pred_original_scale = np.exp(y_pred_log) * correction_factor

# Compare predictions with actual values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted':␣
↪y_pred_original_scale})
print(predictions_df)
```

```
            Actual     Predicted
Date
2017-01-01   21883  19850.333227
2017-02-01   23391  21814.832753
```

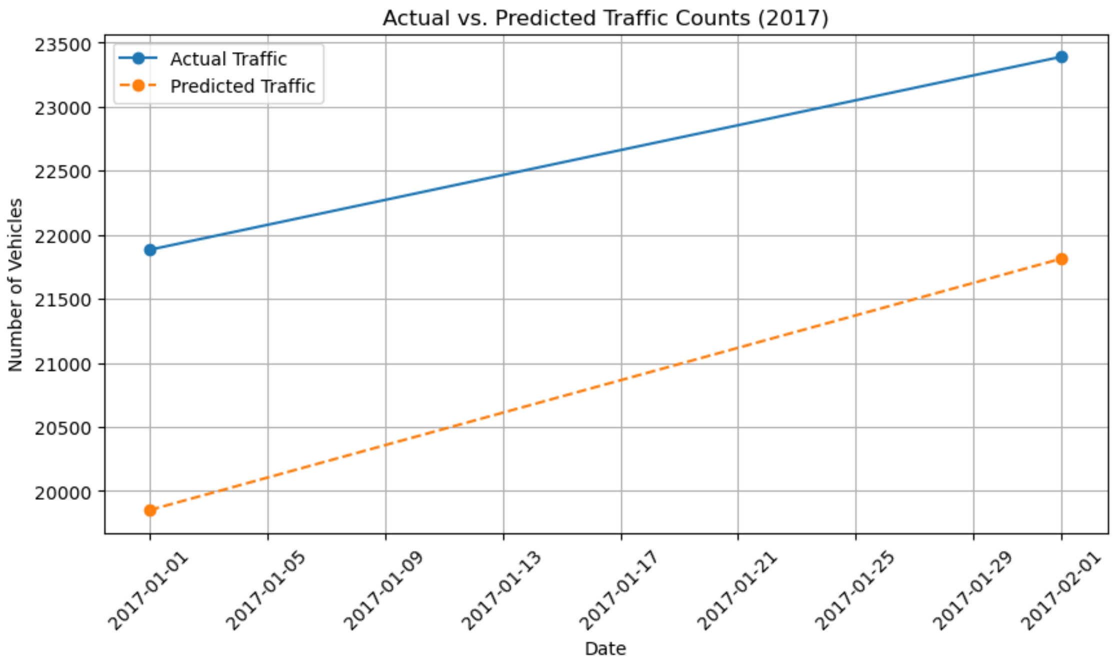
[35]:
```
# Keep only Monthly_Max_Temp
X_selected = df[['Monthly_Max_Temp']]
X_selected = sm.add_constant(X_selected)

# Re-run train-test split
X_train, X_test = X_selected[train_mask], X_selected[~train_mask]

# Refit the log model
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.695
Model:                             OLS   Adj. R-squared:                  0.693
Method:                  Least Squares   F-statistic:                     323.6
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           1.94e-38
Time:                         19:47:28   Log-Likelihood:                 163.94
No. Observations:                  144   AIC:                            -323.9
Df Residuals:                      142   BIC:                            -317.9
Df Model:                            1
Covariance Type:             nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
```

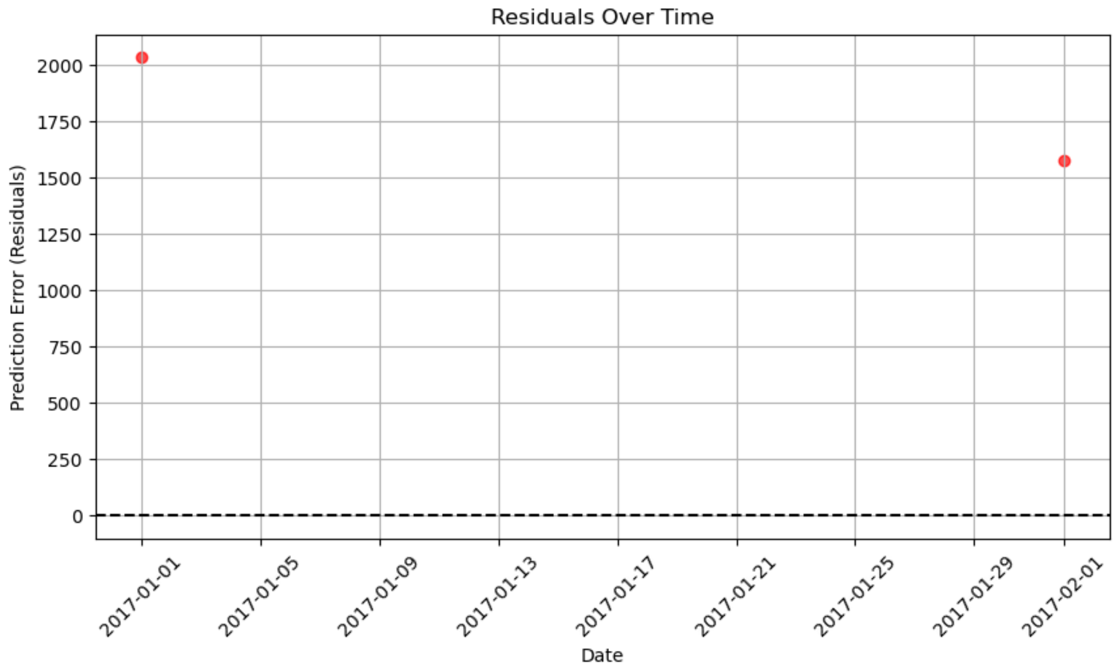
```
----
const                 9.7114      0.020    475.296      0.000       9.671
9.752
Monthly_Max_Temp      0.0067      0.000     17.988      0.000       0.006
0.007
==============================================================================
Omnibus:                        2.277   Durbin-Watson:                   1.029
Prob(Omnibus):                  0.320   Jarque-Bera (JB):                1.883
Skew:                           0.149   Prob(JB):                        0.390
Kurtosis:                       3.474   Cond. No.                         172.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[37]:

```python
# Plot actual vs predicted traffic counts
plt.figure(figsize=(10, 5))
plt.plot(y_test.index, y_test, label="Actual Traffic", marker="o",
 ↪linestyle="-")
plt.plot(y_test.index, y_pred_original_scale, label="Predicted Traffic",
 ↪marker="o", linestyle="dashed")

plt.xlabel("Date")
plt.ylabel("Number of Vehicles")
plt.title("Actual vs. Predicted Traffic Counts (2017)")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



[39]:
```
residuals = y_test - y_pred_original_scale

plt.figure(figsize=(10,5))
plt.scatter(y_test.index, residuals, color='red', alpha=0.7)
plt.axhline(y=0, color='black', linestyle='dashed')
plt.xlabel("Date")
plt.ylabel("Prediction Error (Residuals)")
plt.title("Residuals Over Time")
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



[41]:
```python
# Create a date range for March - December 2017
future_dates = pd.date_range(start="2017-03-01", end="2017-12-01", freq="MS")

# Estimate Monthly_Max_Temp based on previous years (mean temp for each month)
monthly_avg_temp = df.groupby(df.index.month)['Monthly_Max_Temp'].mean()

# Assign estimated Monthly_Max_Temp based on historical averages
future_temps = [monthly_avg_temp[date.month] for date in future_dates]

# Create future DataFrame
future_df = pd.DataFrame({
    'Date': future_dates,
    'Monthly_Max_Temp': future_temps
})

# Set index and add constant for regression
future_df.set_index("Date", inplace=True)
future_df = sm.add_constant(future_df)

print(future_df.head())  # Check the future dataset
```

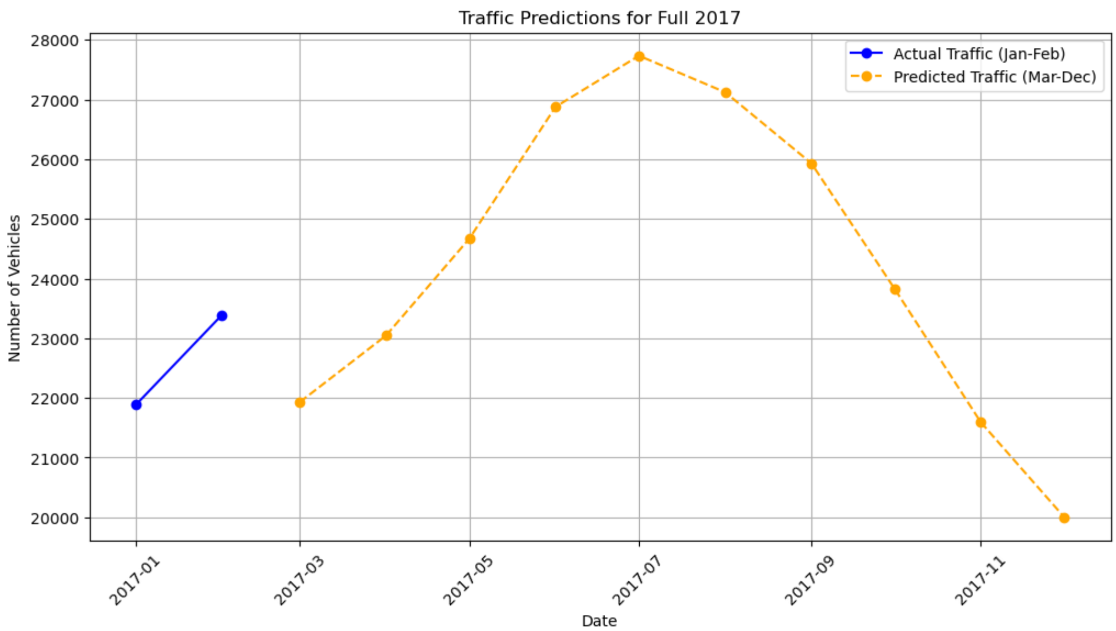
```
            const  Monthly_Max_Temp
Date
2017-03-01    1.0         41.833333
2017-04-01    1.0         49.208333
```

```
2017-05-01    1.0         59.350000
2017-06-01    1.0         72.100000
2017-07-01    1.0         76.766667
```

[43]:
```
# Predict on future data (log scale)
future_pred_log = model_sm_log.predict(future_df)

# Convert back to original scale with correction factor
future_pred_original_scale = np.exp(future_pred_log) * correction_factor

# Create DataFrame for results
future_predictions_df = pd.DataFrame({
    'Predicted_Vehicle_Count': future_pred_original_scale
}, index=future_dates)

print(future_predictions_df)
```

```
            Predicted_Vehicle_Count
2017-03-01             21929.809901
2017-04-01             23044.476880
2017-05-01             24670.410567
2017-06-01             26878.261074
2017-07-01             27734.856813
2017-08-01             27123.305553
2017-09-01             25933.094558
2017-10-01             23825.562431
2017-11-01             21589.710403
2017-12-01             20000.297891
```

[45]:
```
# Combine actual and predicted data
plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label="Actual Traffic (Jan-Feb)", marker="o",␣
 ↪linestyle="-", color="blue")
plt.plot(future_predictions_df.index,␣
 ↪future_predictions_df['Predicted_Vehicle_Count'],
         label="Predicted Traffic (Mar-Dec)", marker="o", linestyle="dashed",␣
 ↪color="orange")

plt.xlabel("Date")
plt.ylabel("Number of Vehicles")
plt.title("Traffic Predictions for Full 2017")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



[47]:
```
# Extract 2016 data
traffic_2016 = df.loc["2016", "Daily Vehicle Count"]
```

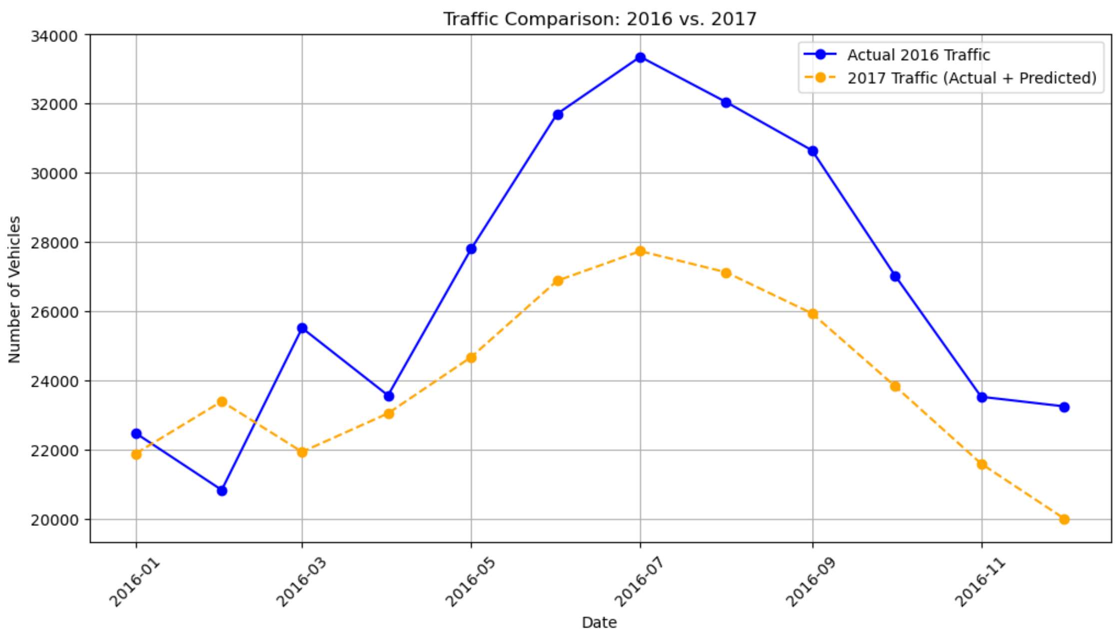
[51]:
```
# Create a DataFrame for comparison
comparison_df = pd.DataFrame({
    '2016 Traffic': traffic_2016.values,  # Actual 2016
    '2017 Traffic': pd.concat([y_test,␣
 ↪future_predictions_df['Predicted_Vehicle_Count']])  # 2017 Actual + Predicted
})

# Set the index (months)
comparison_df.index = pd.date_range(start="2016-01-01", periods=12, freq="MS")

comparison_df.head()
```

[51]:

	2016 Traffic	2017 Traffic
2016-01-01	22470	21883.000000
2016-02-01	20829	23391.000000
2016-03-01	25512	21929.809901
2016-04-01	23563	23044.476880
2016-05-01	27800	24670.410567

[53]:
```
plt.figure(figsize=(12, 6))

# Plot 2016 Actual Data
plt.plot(comparison_df.index, comparison_df['2016 Traffic'],
```

```
        label="Actual 2016 Traffic", marker="o", linestyle="-", color="blue")

# Plot 2017 Actual + Predicted Data
plt.plot(comparison_df.index, comparison_df['2017 Traffic'],
         label="2017 Traffic (Actual + Predicted)", marker="o",␣
 ↪linestyle="dashed", color="orange")

plt.xlabel("Date")
plt.ylabel("Number of Vehicles")
plt.title("Traffic Comparison: 2016 vs. 2017")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



[55]:
```
# Define independent variable (only Monthly Max Temp)
X_selected = df[['Monthly_Max_Temp']]
X_selected = sm.add_constant(X_selected)

# 70/30 train-test split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
 ↪30, random_state=42, shuffle=True)

# Log-transform the target variable
y_train_log = np.log(y_train)
```

```python
# Fit the new model
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.681
Model:                             OLS   Adj. R-squared:                  0.678
Method:                  Least Squares   F-statistic:                     213.4
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           1.51e-26
Time:                         20:04:40   Log-Likelihood:                 122.63
No. Observations:                  102   AIC:                            -241.3
Df Residuals:                      100   BIC:                            -236.0
Df Model:                            1
Covariance Type:             nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const                9.7297      0.023    430.465      0.000       9.685
9.775
Monthly_Max_Temp     0.0063      0.000     14.607      0.000       0.005
0.007
==============================================================================
Omnibus:                        1.153   Durbin-Watson:                   1.847
Prob(Omnibus):                  0.562   Jarque-Bera (JB):                1.047
Skew:                           0.245   Prob(JB):                        0.593
Kurtosis:                       2.927   Cond. No.                         162.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:

Max_Temp_Regular_Gas

March 9, 2025

[1]:
```python
# Importing necessary libraries for time series analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.tsa.api import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statsmodels.api as sm
import seaborn as sns
```

[3]:
```python
# Read in the data
df = pd.read_csv('/Users/helenamabey/Stats_Spring_2025/Congestion Cleaned.csv')
df.head()
```

[3]:
```
         Date Month Year  Daily Vehicle Count  Regular Gas Price Average  \
0  2005-01-01    2005-01                19470                     1.7660
1  2005-02-01    2005-02                21207                     1.8550
2  2005-03-01    2005-03                22943                     2.0825
3  2005-04-01    2005-04                21288                     2.2300
4  2005-05-01    2005-05                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[5]:
```
# Convert 'Date' to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

[7]:
```
# Select independent variables (Adding Regular Gas Price)
X_selected = df[['Monthly_Max_Temp', 'Regular Gas Price Average']]

# Add a constant for the intercept
X_selected = sm.add_constant(X_selected)

# Define the dependent variable
y = df['Daily Vehicle Count']
```

[9]:
```
# 70/30 train-test split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
 ↪30, random_state=42, shuffle=True)

# Log-transform the target variable
y_train_log = np.log(y_train)

# Fit the new model
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.712
Model:                             OLS   Adj. R-squared:                  0.706
Method:                  Least Squares   F-statistic:                     122.1
Date:                 Fri, 07 Mar 2025   Prob (F-statistic):           1.89e-27
Time:                         20:03:29   Log-Likelihood:                 127.78
No. Observations:                  102   AIC:                            -249.6
Df Residuals:                       99   BIC:                            -241.7
Df Model:                            2
Covariance Type:             nonrobust
================================================================================
=============
                                coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-------------
const                         9.8124      0.033    293.738      0.000
9.746       9.879
Monthly_Max_Temp              0.0069      0.000     15.350      0.000
0.006       0.008
```

```
Regular Gas Price Average    -0.0391      0.012     -3.242      0.002
-0.063      -0.015
==============================================================================
Omnibus:                        0.584   Durbin-Watson:                   2.023
Prob(Omnibus):                  0.747   Jarque-Bera (JB):                0.727
Skew:                           0.137   Prob(JB):                        0.695
Kurtosis:                       2.691   Cond. No.                         261.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

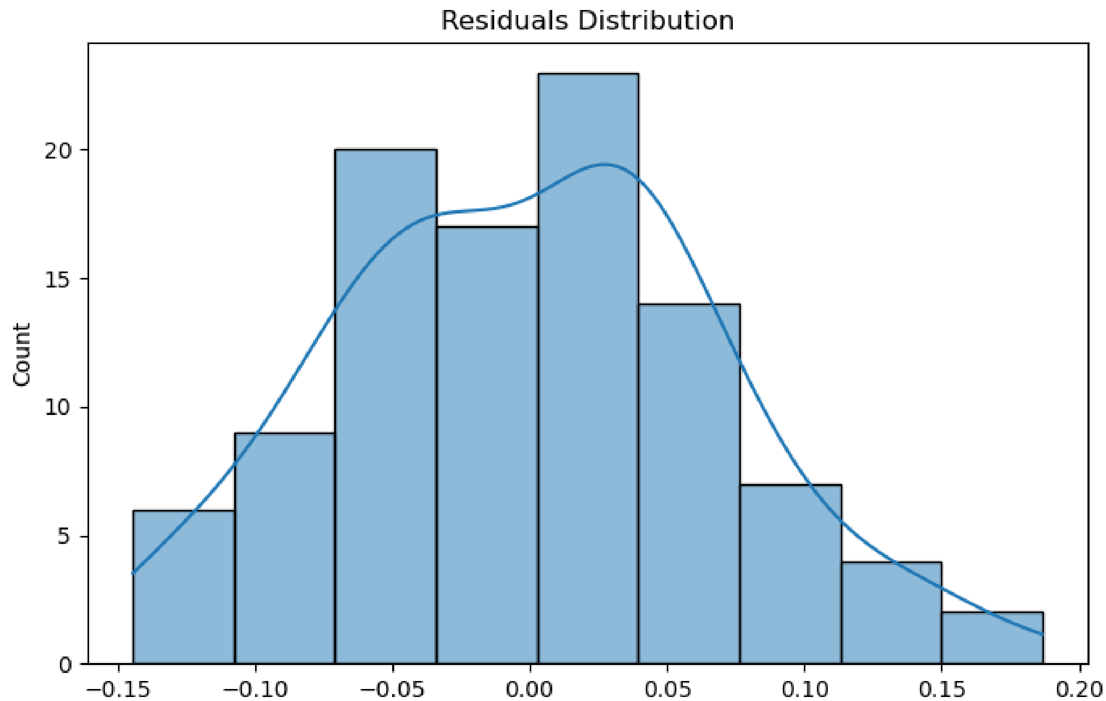
[13]:
```python
# Predict on test data (log scale)
y_pred_log = model_sm_log.predict(X_test)

# Apply correction factor for log transformation bias
correction_factor = np.exp(model_sm_log.resid.var() / 2)
y_pred_original_scale = np.exp(y_pred_log) * correction_factor

# Compare actual vs predicted values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted':
 ↪y_pred_original_scale})
predictions_df.head()
```

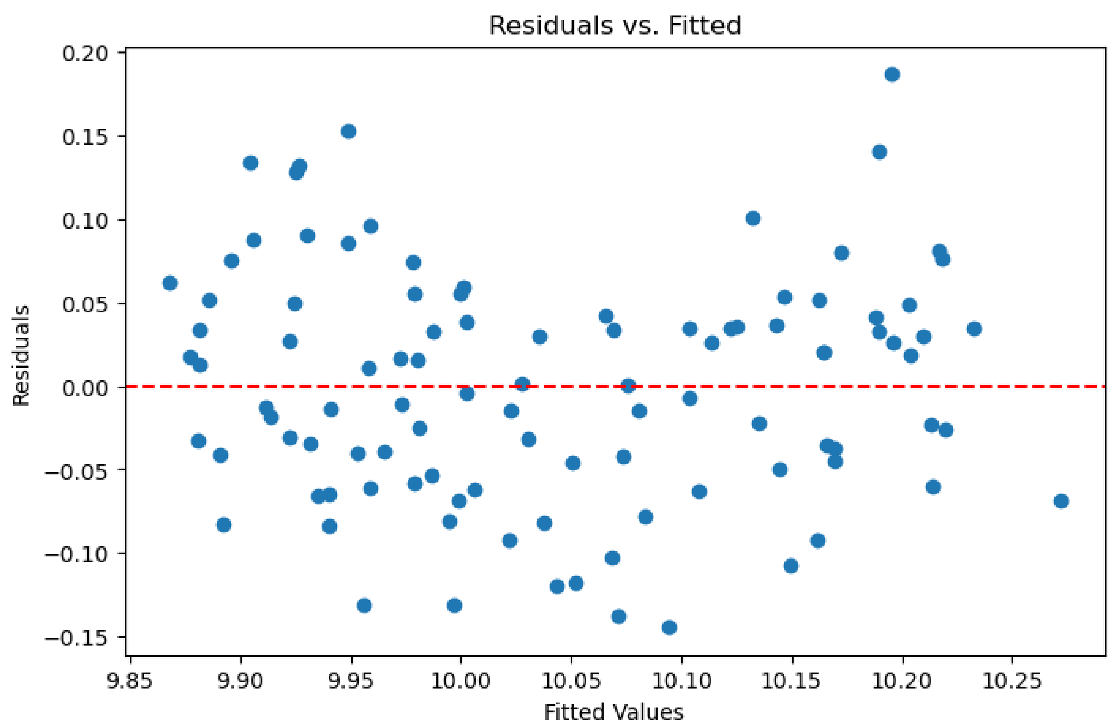
[13]:

Date	Actual	Predicted
2008-10-01	23962	23853.264875
2013-02-01	19101	19654.576378
2007-04-01	22582	23419.099646
2006-08-01	27524	27006.864674
2008-07-01	28666	26780.278233

[15]:
```python
# Get residuals
residuals_log = model_sm_log.resid

# Histogram of residuals
plt.figure(figsize=(8,5))
sns.histplot(residuals_log, kde=True)
plt.title("Residuals Distribution")
plt.show()

# Residuals vs. Fitted Values
plt.figure(figsize=(8,5))
plt.scatter(model_sm_log.fittedvalues, residuals_log)
plt.axhline(y=0, color='r', linestyle='dashed')
plt.xlabel("Fitted Values")
```

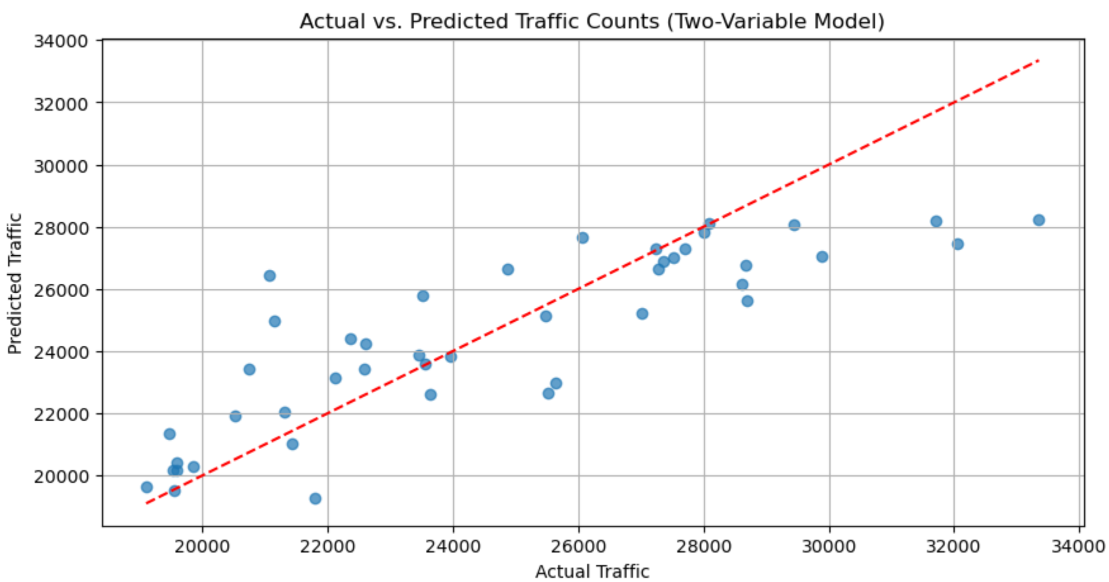
```
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted")
plt.show()
```





[17]:

```
plt.figure(figsize=(10,5))
plt.scatter(y_test, y_pred_original_scale, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color="red", linestyle="dashed")  # 1:1 line
plt.xlabel("Actual Traffic")
plt.ylabel("Predicted Traffic")
plt.title("Actual vs. Predicted Traffic Counts (Two-Variable Model)")
plt.grid()
plt.show()
```



[21]:

```
# Create future dates for March - December 2017
future_dates = pd.date_range(start="2017-03-01", end="2017-12-01", freq="MS")

# Estimate Monthly_Max_Temp and Regular Gas Price Average using historical means
monthly_avg_temp = df.groupby(df.index.month)['Monthly_Max_Temp'].mean()
monthly_avg_gas = df.groupby(df.index.month)['Regular Gas Price Average'].mean()

# Assign estimated values based on historical averages
future_temps = [monthly_avg_temp[date.month] for date in future_dates]
future_gas_prices = [monthly_avg_gas[date.month] for date in future_dates]

# Create future DataFrame
future_df = pd.DataFrame({
    'Monthly_Max_Temp': future_temps,
    'Regular Gas Price Average': future_gas_prices
```

```
}, index=future_dates)

# Add constant for regression model
future_df = sm.add_constant(future_df)

future_df.head()  # Check future data
```

[21]:

	const	Monthly_Max_Temp	Regular Gas Price Average
2017-03-01	1.0	41.833333	2.724792
2017-04-01	1.0	49.208333	2.880000
2017-05-01	1.0	59.350000	3.014167
2017-06-01	1.0	72.100000	3.050000
2017-07-01	1.0	76.766667	3.038458

[23]:
```
# Predict on future data (March - December 2017)
future_pred_log = model_sm_log.predict(future_df)

# Convert back to original scale
future_pred_original_scale = np.exp(future_pred_log) * correction_factor

# Store predictions
future_predictions_df = pd.DataFrame({'Predicted_Vehicle_Count':␣
↪future_pred_original_scale}, index=future_dates)

future_predictions_df
```

[23]:

	Predicted_Vehicle_Count
2017-03-01	21942.236112
2017-04-01	22945.071702
2017-05-01	24475.123562
2017-06-01	26682.499022
2017-07-01	27565.842751
2017-08-01	26953.912706
2017-09-01	25736.434203
2017-10-01	23764.819566
2017-11-01	21651.298980
2017-12-01	20153.574913

[35]:
```
# Extract 2016 actual traffic (Jan-Dec)
traffic_2016 = df.loc["2016", "Daily Vehicle Count"]

# Extract ONLY January & February 2017 actual traffic counts
traffic_2017_actual = df.loc["2017-01":"2017-02", "Daily Vehicle Count"]

# Extract 2017 predicted traffic (Mar-Dec only)
traffic_2017_predicted = future_predictions_df['Predicted_Vehicle_Count']
```

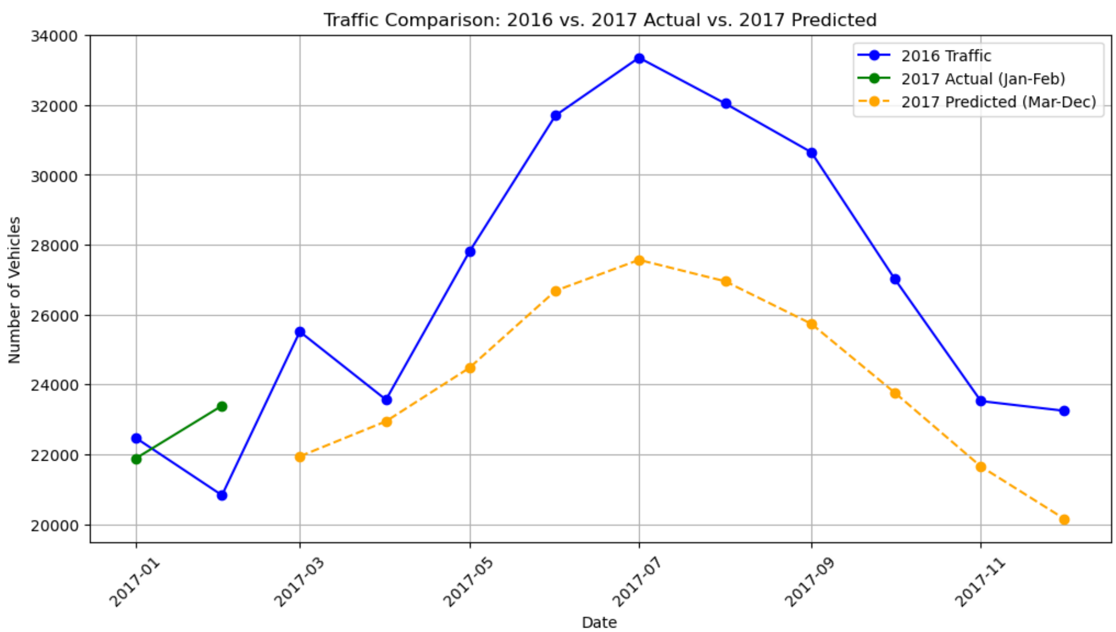
```
# Create a full index for Jan-Dec 2017
full_2017_index = pd.date_range(start="2017-01-01", periods=12, freq="MS")
```

[37]:
```
# Create an empty DataFrame with the full 2017 index
comparison_df = pd.DataFrame(index=full_2017_index)

# Assign 2016 actual traffic (aligned with Jan-Dec)
comparison_df['2016 Traffic'] = traffic_2016.values

# Assign 2017 actual traffic (only for Jan-Feb)
comparison_df.loc["2017-01-01":"2017-02-01", '2017 Actual'] =␣
↪traffic_2017_actual.values

# Assign 2017 predicted traffic (only for Mar-Dec)
comparison_df.loc["2017-03-01":"2017-12-01", '2017 Predicted'] =␣
↪traffic_2017_predicted.values

comparison_df
```

[37]:
```
            2016 Traffic  2017 Actual  2017 Predicted
2017-01-01         22470      21883.0             NaN
2017-02-01         20829      23391.0             NaN
2017-03-01         25512          NaN    21942.236112
2017-04-01         23563          NaN    22945.071702
2017-05-01         27800          NaN    24475.123562
2017-06-01         31702          NaN    26682.499022
2017-07-01         33354          NaN    27565.842751
2017-08-01         32044          NaN    26953.912706
2017-09-01         30647          NaN    25736.434203
2017-10-01         27020          NaN    23764.819566
2017-11-01         23524          NaN    21651.298980
2017-12-01         23246          NaN    20153.574913
```

[39]:
```
plt.figure(figsize=(12, 6))

# Plot actual 2016 traffic
plt.plot(comparison_df.index, comparison_df['2016 Traffic'], label="2016␣
↪Traffic", marker="o", linestyle="-", color="blue")

# Plot actual 2017 traffic (Jan-Feb)
plt.plot(comparison_df.index[:2], comparison_df['2017 Actual'].dropna(),␣
↪label="2017 Actual (Jan-Feb)", marker="o", linestyle="-", color="green")

# Plot predicted 2017 traffic (Mar-Dec)
plt.plot(comparison_df.index[2:], comparison_df['2017 Predicted'].dropna(),␣
↪label="2017 Predicted (Mar-Dec)", marker="o", linestyle="dashed",␣
↪color="orange")
```

```python
plt.xlabel("Date")
plt.ylabel("Number of Vehicles")
plt.title("Traffic Comparison: 2016 vs. 2017 Actual vs. 2017 Predicted")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



[ ]:

```python

```

Dummy_data_quadratic

March 9, 2025

[1]:
```python
# Importing necessary libraries for time series analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
from statsmodels.tsa.api import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statsmodels.api as sm
import seaborn as sns
```

[3]:
```python
# Read in the data
df = pd.read_csv('/Users/helenamabey/Stats_Spring_2025/Congestion Cleaned.csv')
df.head()
```

[3]:
```
         Date Month Year  Daily Vehicle Count  Regular Gas Price Average  \
0  2005-01-01    2005-01                19470                     1.7660
1  2005-02-01    2005-02                21207                     1.8550
2  2005-03-01    2005-03                22943                     2.0825
3  2005-04-01    2005-04                21288                     2.2300
4  2005-05-01    2005-05                23505                     2.1540

   Midgrade Gas Price Average  Premium Gas Price Average  Monthly_Max_Temp  \
0                      1.8760                     1.9800              32.3
1                      1.9650                     2.0650              33.9
2                      2.1900                     2.2875              39.5
3                      2.3425                     2.4450              49.9
4                      2.2640                     2.3640              62.1

   Monthly_Min_Temp  Monthly_Sum_Precipitation  Monthly_Snowfall
0              12.7                       2.29              36.0
1              10.5                       2.14              37.5
2              17.5                       1.49              20.0
3              24.0                       2.24              20.0
4              30.5                       1.01               2.0
```

[5]:
```python
# Convert 'Date' to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

[7]:
```python
# Define function to assign seasons
def assign_season(month):
    if month in [12, 1, 2]:
        return "Winter"
    elif month in [3, 4, 5]:
        return "Spring"
    elif month in [6, 7, 8]:
        return "Summer"
    else:
        return "Fall"

# Create 'Season' column
df['Season'] = df.index.month.map(assign_season)

# Convert 'Season' to dummy variables (one-hot encoding)
df = pd.get_dummies(df, columns=['Season'], drop_first=True)

df.head() # Verify dummy columns are added
```

[7]:
```
           Month Year  Daily Vehicle Count  Regular Gas Price Average  \
Date
2005-01-01    2005-01                19470                     1.7660
2005-02-01    2005-02                21207                     1.8550
2005-03-01    2005-03                22943                     2.0825
2005-04-01    2005-04                21288                     2.2300
2005-05-01    2005-05                23505                     2.1540

            Midgrade Gas Price Average  Premium Gas Price Average  \
Date
2005-01-01                      1.8760                     1.9800
2005-02-01                      1.9650                     2.0650
2005-03-01                      2.1900                     2.2875
2005-04-01                      2.3425                     2.4450
2005-05-01                      2.2640                     2.3640

            Monthly_Max_Temp  Monthly_Min_Temp  Monthly_Sum_Precipitation  \
Date
2005-01-01              32.3              12.7                       2.29
2005-02-01              33.9              10.5                       2.14
2005-03-01              39.5              17.5                       1.49
2005-04-01              49.9              24.0                       2.24
2005-05-01              62.1              30.5                       1.01
```

```
            Monthly_Snowfall  Season_Spring  Season_Summer  Season_Winter
Date
2005-01-01              36.0              0              0              1
2005-02-01              37.5              0              0              1
2005-03-01              20.0              1              0              0
2005-04-01              20.0              1              0              0
2005-05-01               2.0              1              0              0
```

[9]:
```
# Add quadratic transformation for Monthly_Max_Temp
df['Monthly_Max_Temp_Squared'] = df['Monthly_Max_Temp'] ** 2
df.head()
```

[9]:
```
           Month Year  Daily Vehicle Count  Regular Gas Price Average  \
Date
2005-01-01    2005-01                19470                     1.7660
2005-02-01    2005-02                21207                     1.8550
2005-03-01    2005-03                22943                     2.0825
2005-04-01    2005-04                21288                     2.2300
2005-05-01    2005-05                23505                     2.1540

            Midgrade Gas Price Average  Premium Gas Price Average  \
Date
2005-01-01                      1.8760                     1.9800
2005-02-01                      1.9650                     2.0650
2005-03-01                      2.1900                     2.2875
2005-04-01                      2.3425                     2.4450
2005-05-01                      2.2640                     2.3640

            Monthly_Max_Temp  Monthly_Min_Temp  Monthly_Sum_Precipitation  \
Date
2005-01-01              32.3              12.7                       2.29
2005-02-01              33.9              10.5                       2.14
2005-03-01              39.5              17.5                       1.49
2005-04-01              49.9              24.0                       2.24
2005-05-01              62.1              30.5                       1.01

            Monthly_Snowfall  Season_Spring  Season_Summer  Season_Winter  \
Date
2005-01-01              36.0              0              0              1
2005-02-01              37.5              0              0              1
2005-03-01              20.0              1              0              0
2005-04-01              20.0              1              0              0
2005-05-01               2.0              1              0              0

            Monthly_Max_Temp_Squared
Date
2005-01-01                   1043.29
```

```
2005-02-01                    1149.21
2005-03-01                    1560.25
2005-04-01                    2490.01
2005-05-01                    3856.41
```

[13]:
```
# Identify available season columns (Pandas may have dropped one)
season_dummies = [col for col in df.columns if col.startswith("Season_")]

# Check what was created
print("Available season columns:", season_dummies)
```

```
Available season columns: ['Season_Spring', 'Season_Summer', 'Season_Winter']
```

[15]:
```
# Drop Winter as baseline (or another if needed)
if "Season_Winter" in season_dummies:
    season_dummies.remove("Season_Winter")
```

[17]:
```
# Select independent variables dynamically
X = df[['Monthly_Max_Temp', 'Monthly_Max_Temp_Squared', 'Regular Gas Price␣
 ↪Average'] + season_dummies]

# Add constant for regression
X = sm.add_constant(X)

# Check the final DataFrame
print(X.head())  # Verify correct columns are included
```

```
            const  Monthly_Max_Temp  Monthly_Max_Temp_Squared  \
Date
2005-01-01    1.0              32.3                   1043.29
2005-02-01    1.0              33.9                   1149.21
2005-03-01    1.0              39.5                   1560.25
2005-04-01    1.0              49.9                   2490.01
2005-05-01    1.0              62.1                   3856.41

            Regular Gas Price Average  Season_Spring  Season_Summer
Date
2005-01-01                     1.7660              0              0
2005-02-01                     1.8550              0              0
2005-03-01                     2.0825              1              0
2005-04-01                     2.2300              1              0
2005-05-01                     2.1540              1              0
```

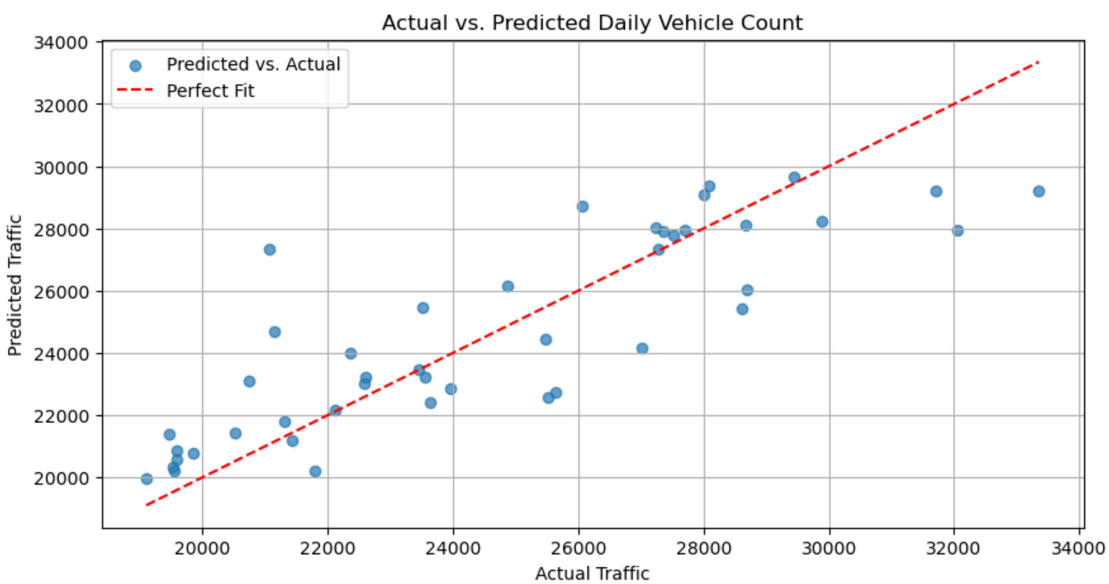
[19]:
```
# Define dependent variable
y = df['Daily Vehicle Count']

# 70/30 train-test split
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=42, shuffle=True)

# Log-transform the target variable
y_train_log = np.log(y_train)

# Fit the model
model_sm_log = sm.OLS(y_train_log, X_train).fit()

# Print model summary
print(model_sm_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Daily Vehicle Count   R-squared:                       0.750
Model:                             OLS   Adj. R-squared:                  0.737
Method:                  Least Squares   F-statistic:                     57.49
Date:                 Sat, 08 Mar 2025   Prob (F-statistic):           2.30e-27
Time:                         08:40:42   Log-Likelihood:                 135.01
No. Observations:                  102   AIC:                            -258.0
Df Residuals:                       96   BIC:                            -242.3
Df Model:                            5
Covariance Type:             nonrobust
================================================================================
=============
                                coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-------------
const                        10.0479      0.098    102.807      0.000
9.854      10.242
Monthly_Max_Temp             -0.0035      0.004     -0.818      0.415
-0.012       0.005
Monthly_Max_Temp_Squared   9.588e-05   4.52e-05      2.121      0.036
6.16e-06       0.000
Regular Gas Price Average    -0.0376      0.012     -3.202      0.002
-0.061      -0.014
Season_Spring                 0.0257      0.019      1.379      0.171
-0.011       0.063
Season_Summer                 0.0354      0.033      1.078      0.284
-0.030       0.101
==============================================================================
Omnibus:                        1.161   Durbin-Watson:                   2.248
Prob(Omnibus):                  0.560   Jarque-Bera (JB):                1.238
Skew:                           0.236   Prob(JB):                        0.539
Kurtosis:                       2.740   Cond. No.                     4.90e+04
==============================================================================
```

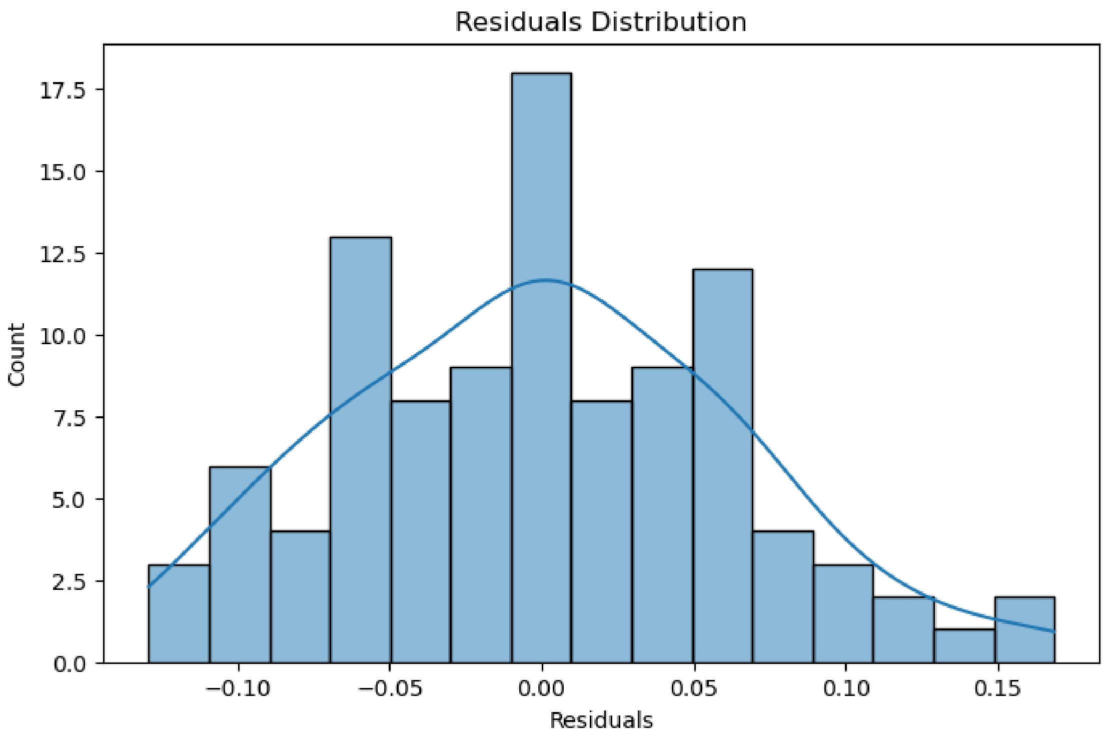
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.9e+04. This might indicate that there are strong multicollinearity or other numerical problems.

[21]:
```
# Predict on test data (log scale)
y_pred_log = model_sm_log.predict(X_test)

# Convert back from log-scale
correction_factor = np.exp(model_sm_log.resid.var() / 2)
y_pred_original_scale = np.exp(y_pred_log) * correction_factor

# Create plot
plt.figure(figsize=(10,5))
plt.scatter(y_test, y_pred_original_scale, alpha=0.7, label="Predicted vs.␣
↪Actual")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
↪color="red", linestyle="dashed", label="Perfect Fit")
plt.xlabel("Actual Traffic")
plt.ylabel("Predicted Traffic")
plt.title("Actual vs. Predicted Daily Vehicle Count")
plt.legend()
plt.grid()
plt.show()
```



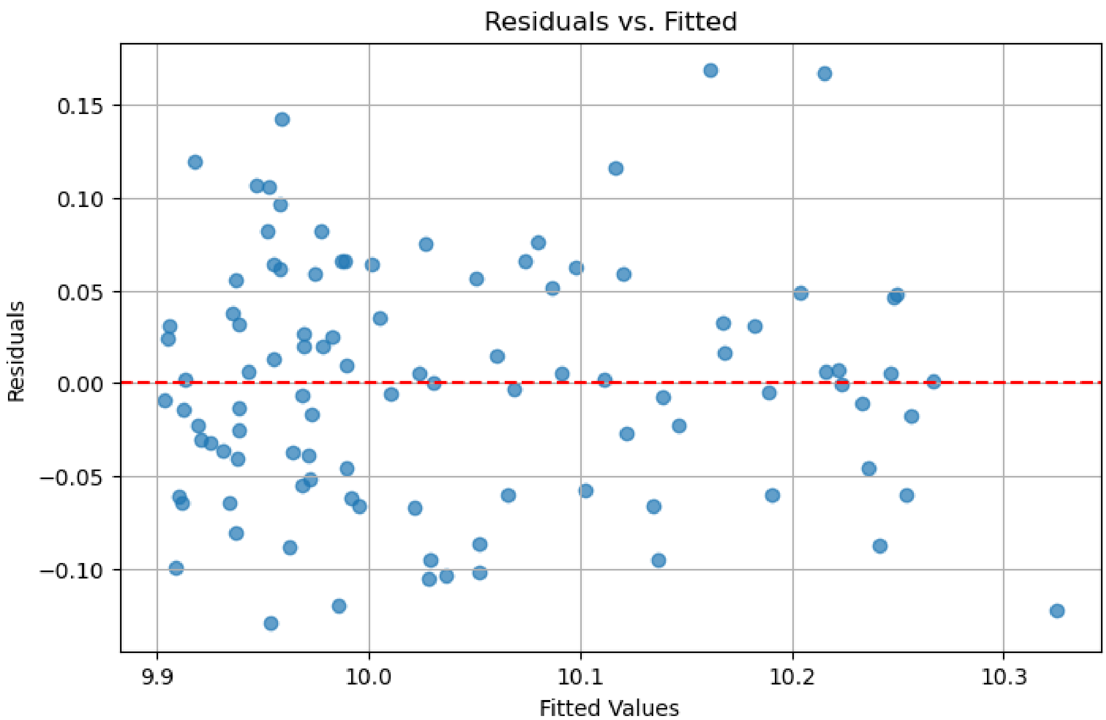
[29]:

```
residuals = model_sm_log.resid

# Plot residuals distribution
plt.figure(figsize=(8,5))
sns.histplot(residuals, kde=True, bins=15)
plt.title("Residuals Distribution")
plt.xlabel("Residuals")
plt.show()
```

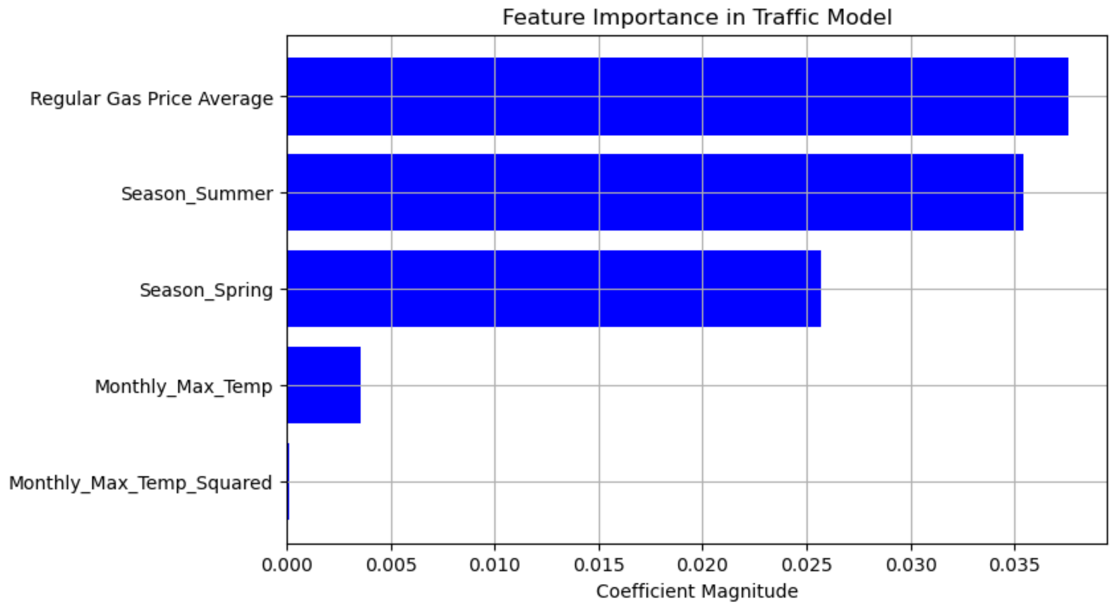


[31]:

```
plt.figure(figsize=(8,5))
plt.scatter(model_sm_log.fittedvalues, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='dashed')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted")
plt.grid()
plt.show()
```



[33]:
```
coefs = model_sm_log.params[1:]  # Exclude constant
coefs = coefs.abs().sort_values()  # Sort by magnitude

plt.figure(figsize=(8,5))
plt.barh(coefs.index, coefs.values, color="blue")
plt.xlabel("Coefficient Magnitude")
plt.title("Feature Importance in Traffic Model")
plt.grid()
plt.show()
```



[35]:
```
# Create date range for March - December 2017
future_dates = pd.date_range(start="2017-03-01", end="2017-12-01", freq="MS")

# Estimate Monthly_Max_Temp & Regular Gas Price based on historical monthly␣
↪averages
monthly_avg_temp = df.groupby(df.index.month)['Monthly_Max_Temp'].mean()
monthly_avg_gas = df.groupby(df.index.month)['Regular Gas Price Average'].mean()

# Assign estimated values for future months
future_temps = [monthly_avg_temp[date.month] for date in future_dates]
future_gas_prices = [monthly_avg_gas[date.month] for date in future_dates]

# Create DataFrame for future data
future_df = pd.DataFrame({
    'Monthly_Max_Temp': future_temps,
    'Regular Gas Price Average': future_gas_prices
}, index=future_dates)

# Add the quadratic term for Monthly_Max_Temp
future_df['Monthly_Max_Temp_Squared'] = future_df['Monthly_Max_Temp'] ** 2

# Assign seasons and create dummy variables
def assign_season(month):
    if month in [12, 1, 2]:
        return "Winter"
    elif month in [3, 4, 5]:
        return "Spring"
```

```
    elif month in [6, 7, 8]:
        return "Summer"
    else:
        return "Fall"

future_df['Season'] = future_df.index.month.map(assign_season)

# One-hot encode seasons (ensure alignment with training data)
future_df = pd.get_dummies(future_df, columns=['Season'], drop_first=False)

# Drop Winter as baseline if it exists
if "Season_Winter" in future_df.columns:
    future_df.drop(columns=["Season_Winter"], inplace=True)

# Ensure same feature columns as the model
future_df = sm.add_constant(future_df)

future_df.head()  # Verify the structure
```

[35]:

```
            const  Monthly_Max_Temp  Regular Gas Price Average  \
2017-03-01    1.0         41.833333                   2.724792
2017-04-01    1.0         49.208333                   2.880000
2017-05-01    1.0         59.350000                   3.014167
2017-06-01    1.0         72.100000                   3.050000
2017-07-01    1.0         76.766667                   3.038458

            Monthly_Max_Temp_Squared  Season_Fall  Season_Spring  \
2017-03-01               1750.027778            0              1
2017-04-01               2421.460069            0              1
2017-05-01               3522.422500            0              1
2017-06-01               5198.410000            0              0
2017-07-01               5893.121111            0              0

            Season_Summer
2017-03-01              0
2017-04-01              0
2017-05-01              0
2017-06-01              1
2017-07-01              1
```

[39]:

```
print("Model Features:", X_train.columns)
print("Future Data Features:", future_df.columns)
```

```
Model Features: Index(['const', 'Monthly_Max_Temp', 'Monthly_Max_Temp_Squared',
       'Regular Gas Price Average', 'Season_Spring', 'Season_Summer'],
      dtype='object')
Future Data Features: Index(['const', 'Monthly_Max_Temp', 'Regular Gas Price
```

```
Average',
       'Monthly_Max_Temp_Squared', 'Season_Fall', 'Season_Spring',
       'Season_Summer'],
      dtype='object')
```

[41]:
```python
# Ensure all expected season columns exist in future_df
expected_season_dummies = ['Season_Spring', 'Season_Summer']  # The ones in the␣
 ↪model

# Remove extra season columns if they exist
for col in future_df.columns:
    if col.startswith("Season_") and col not in expected_season_dummies:
        future_df.drop(columns=[col], inplace=True)

# Add missing season columns if they are not present
for col in expected_season_dummies:
    if col not in future_df.columns:
        future_df[col] = 0  # Assign 0 for missing season

# Ensure final feature set matches model expectations
future_df = future_df[X_train.columns]

# Print final check
print("Updated Future Data Features:", future_df.columns)
```

```
Updated Future Data Features: Index(['const', 'Monthly_Max_Temp',
'Monthly_Max_Temp_Squared',
       'Regular Gas Price Average', 'Season_Spring', 'Season_Summer'],
      dtype='object')
```

[43]:
```python
# Ensure all expected season columns exist in future_df
expected_season_dummies = ['Season_Spring', 'Season_Summer']  # The ones in the␣
 ↪model

# Remove extra season columns if they exist
for col in future_df.columns:
    if col.startswith("Season_") and col not in expected_season_dummies:
        future_df.drop(columns=[col], inplace=True)

# Add missing season columns if they are not present
for col in expected_season_dummies:
    if col not in future_df.columns:
        future_df[col] = 0  # Assign 0 for missing season

# Ensure final feature set matches model expectations
future_df = future_df[X_train.columns]
```

```
# Print final check
print("Updated Future Data Features:", future_df.columns)
```

```
Updated Future Data Features: Index(['const', 'Monthly_Max_Temp',
'Monthly_Max_Temp_Squared',
       'Regular Gas Price Average', 'Season_Spring', 'Season_Summer'],
      dtype='object')
```

[45]:
```
# Predict on corrected future data
future_pred_log = model_sm_log.predict(future_df)

# Convert back from log scale
correction_factor = np.exp(model_sm_log.resid.var() / 2)
future_pred_original_scale = np.exp(future_pred_log) * correction_factor

# Store predictions
future_predictions_df = pd.DataFrame({'Predicted_Vehicle_Count':␣
 ↪future_pred_original_scale}, index=future_dates)

print(future_predictions_df)  # Check final predictions
```

```
            Predicted_Vehicle_Count
2017-03-01             21887.750817
2017-04-01             22612.009212
2017-05-01             24125.269690
2017-06-01             27314.468060
2017-07-01             28732.080849
2017-08-01             27723.929848
2017-09-01             25037.186087
2017-10-01             22768.759211
2017-11-01             21175.313216
2017-12-01             20604.289125
```

[47]:
```
# Extract actual 2016 traffic (Jan-Dec)
traffic_2016 = df.loc["2016", "Daily Vehicle Count"]

# Extract actual 2017 traffic (Jan-Feb)
traffic_2017_actual = df.loc["2017-01":"2017-02", "Daily Vehicle Count"]

# Create full 2017 index (Jan-Dec)
full_2017_index = pd.date_range(start="2017-01-01", periods=12, freq="MS")

# Create a DataFrame with full 2017 index
comparison_df = pd.DataFrame(index=full_2017_index)

# Assign 2016 actual traffic (aligned with Jan-Dec)
comparison_df['2016 Traffic'] = traffic_2016.values[:12]
```

```
# Assign 2017 actual traffic (only Jan-Feb)
comparison_df.loc["2017-01-01":"2017-02-01", '2017 Actual'] =␣
↪traffic_2017_actual.values

# Assign 2017 predicted traffic (only Mar-Dec)
comparison_df.loc["2017-03-01":"2017-12-01", '2017 Predicted'] =␣
↪future_predictions_df['Predicted_Vehicle_Count'].values

# Print final comparison table
comparison_df
```

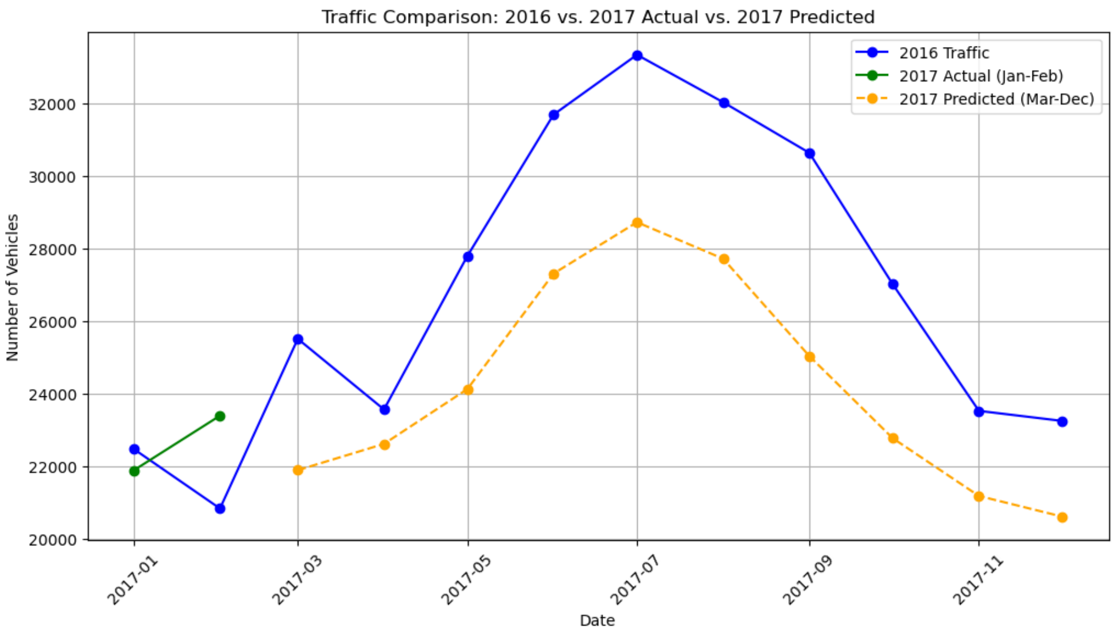
[47]:

	2016 Traffic	2017 Actual	2017 Predicted
2017-01-01	22470	21883.0	NaN
2017-02-01	20829	23391.0	NaN
2017-03-01	25512	NaN	21887.750817
2017-04-01	23563	NaN	22612.009212
2017-05-01	27800	NaN	24125.269690
2017-06-01	31702	NaN	27314.468060
2017-07-01	33354	NaN	28732.080849
2017-08-01	32044	NaN	27723.929848
2017-09-01	30647	NaN	25037.186087
2017-10-01	27020	NaN	22768.759211
2017-11-01	23524	NaN	21175.313216
2017-12-01	23246	NaN	20604.289125

[49]:

```
plt.figure(figsize=(12, 6))

# Plot actual 2016 traffic
plt.plot(comparison_df.index, comparison_df['2016 Traffic'], label="2016␣
↪Traffic", marker="o", linestyle="-", color="blue")

# Plot actual 2017 traffic (Jan-Feb)
plt.plot(comparison_df.index[:2], comparison_df['2017 Actual'].dropna(),␣
↪label="2017 Actual (Jan-Feb)", marker="o", linestyle="-", color="green")

# Plot predicted 2017 traffic (Mar-Dec)
plt.plot(comparison_df.index[2:], comparison_df['2017 Predicted'].dropna(),␣
↪label="2017 Predicted (Mar-Dec)", marker="o", linestyle="dashed",␣
↪color="orange")

plt.xlabel("Date")
plt.ylabel("Number of Vehicles")
plt.title("Traffic Comparison: 2016 vs. 2017 Actual vs. 2017 Predicted")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
```

```
plt.show()
```



Traffic Comparison: 2016 vs. 2017 Actual vs. 2017 Predicted
2016 Traffic
2017 Actual (Jan-Feb)
2017 Predicted (Mar-Dec)
Number of Vehicles
20000
22000
24000
26000
28000
30000
32000
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
Date

[ ]: