In [1]:

```
!pip install numpy
!pip install pandas
!pip install seaborn
```

```
Requirement already satisfied: numpy in c:\users\agrocel\anaconda3\lib\site-
packages (1.24.3)
Requirement already satisfied: pandas in c:\users\agrocel\anaconda3\lib\site
-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\agrocel\an
aconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\agrocel\anaconda3\li
b\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\agrocel\anaconda3
\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\agrocel\anaconda3\l
ib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\agrocel\anaconda3\lib\si
te-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: seaborn in c:\users\agrocel\anaconda3\lib\sit
e-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\agrocel\anac
onda3\lib\site-packages (from seaborn) (1.24.3)
Requirement already satisfied: pandas>=0.25 in c:\users\agrocel\anaconda3\li
b\site-packages (from seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\agrocel\a
naconda3\lib\site-packages (from seaborn) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\agrocel\anaconda
3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\agrocel\anaconda3\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\agrocel\anacond
a3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\agrocel\anacond
a3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\agrocel\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\agrocel\anaconda3\l
ib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\agrocel\ana
conda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\agrocel\anac
onda3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\agrocel\anaconda3\li
b\site-packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\agrocel\anaconda3
\lib\site-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\agrocel\anaconda3\lib\si
te-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn)
(1.16.0)
```

In [2]:

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

In [3]:

```
df = pd.read_excel(r'Agrocel_Jupyter.xlsx')
```

In [4]:

```
df
```

Out[4]:

	Date	Sulphur Rate
0	2021-04-01	23.31
1	2021-04-02	NaN
2	2021-04-03	22.96
3	2021-04-04	NaN
4	2021-04-05	NaN
...	...	...
960	2023-11-17	NaN
961	2023-11-18	NaN
962	2023-11-19	NaN
963	2023-11-20	NaN
964	2023-11-21	NaN

965 rows × 2 columns

In [5]:

```
df.columns
```

Out[5]:

```
Index([' Date', 'Sulphur Rate'], dtype='object')
```

In [6]:

```
df = df.rename(columns={' Date': 'Date'})
```

In [7]:

```
from sklearn.impute import KNNImputer

# Initialize the KNN imputer
k = 5  # Number of neighbors to consider
imputer = KNNImputer(n_neighbors=k)

df_subset = df[['Sulphur Rate']]
df_imputed = imputer.fit_transform(df_subset)
df['Sulphur Rate'] = df_imputed
```

In [8]:

```
df['Date'] = pd.to_datetime(df['Date'])
```

In [9]:

```
df
```

Out[9]:

	Date	Sulphur Rate
0	2021-04-01	23.310000
1	2021-04-02	24.875247
2	2021-04-03	22.960000
3	2021-04-04	24.875247
4	2021-04-05	24.875247
...	...	...
960	2023-11-17	24.875247
961	2023-11-18	24.875247
962	2023-11-19	24.875247
963	2023-11-20	24.875247
964	2023-11-21	24.875247

965 rows × 2 columns

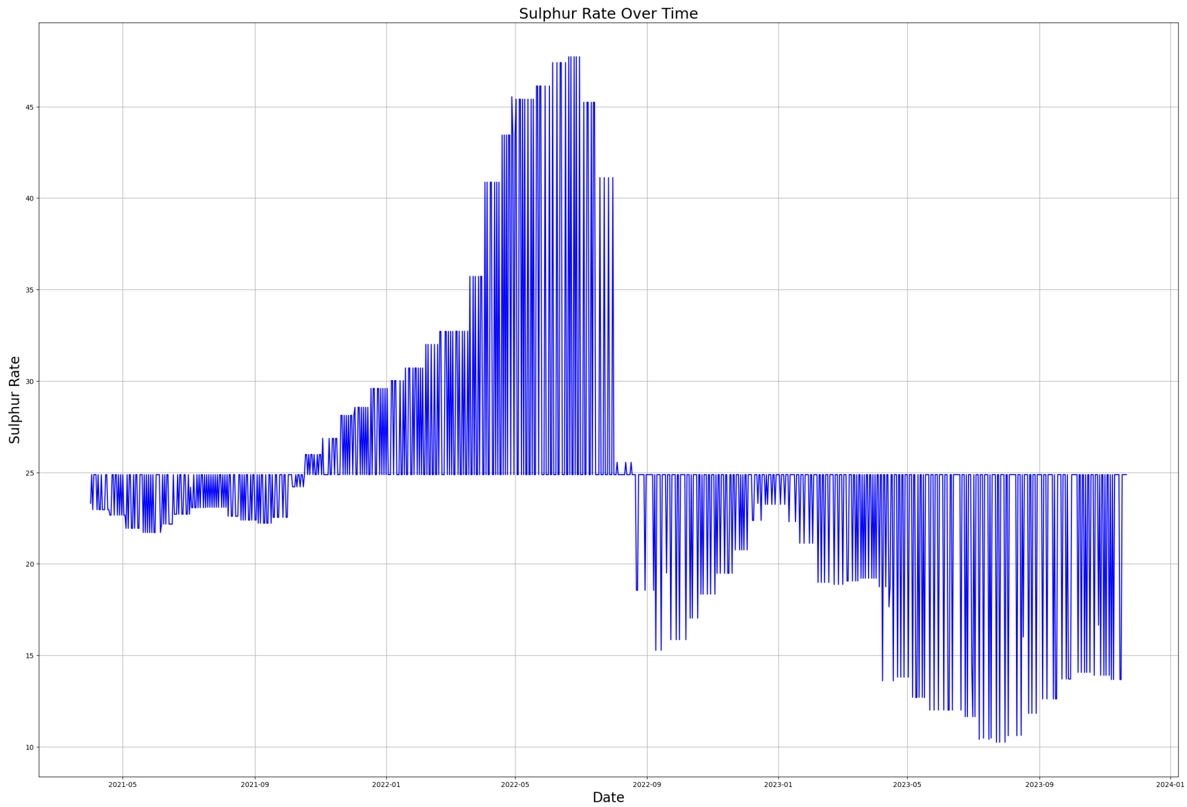
In [10]:

```
print(df.dtypes)
```

```
Date            datetime64[ns]
Sulphur Rate           float64
dtype: object
```

In [11]:

```
import matplotlib.pyplot as plt
import pandas as pd

# Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Plot the time series
plt.figure(figsize=(30, 20))
plt.plot(df['Date'], df['Sulphur Rate'], linestyle='-', color='b')
plt.title('Sulphur Rate Over Time', fontsize=22)
plt.xlabel('Date', fontsize=20)
plt.ylabel('Sulphur Rate', fontsize=20)
plt.grid(True)
plt.show()
```



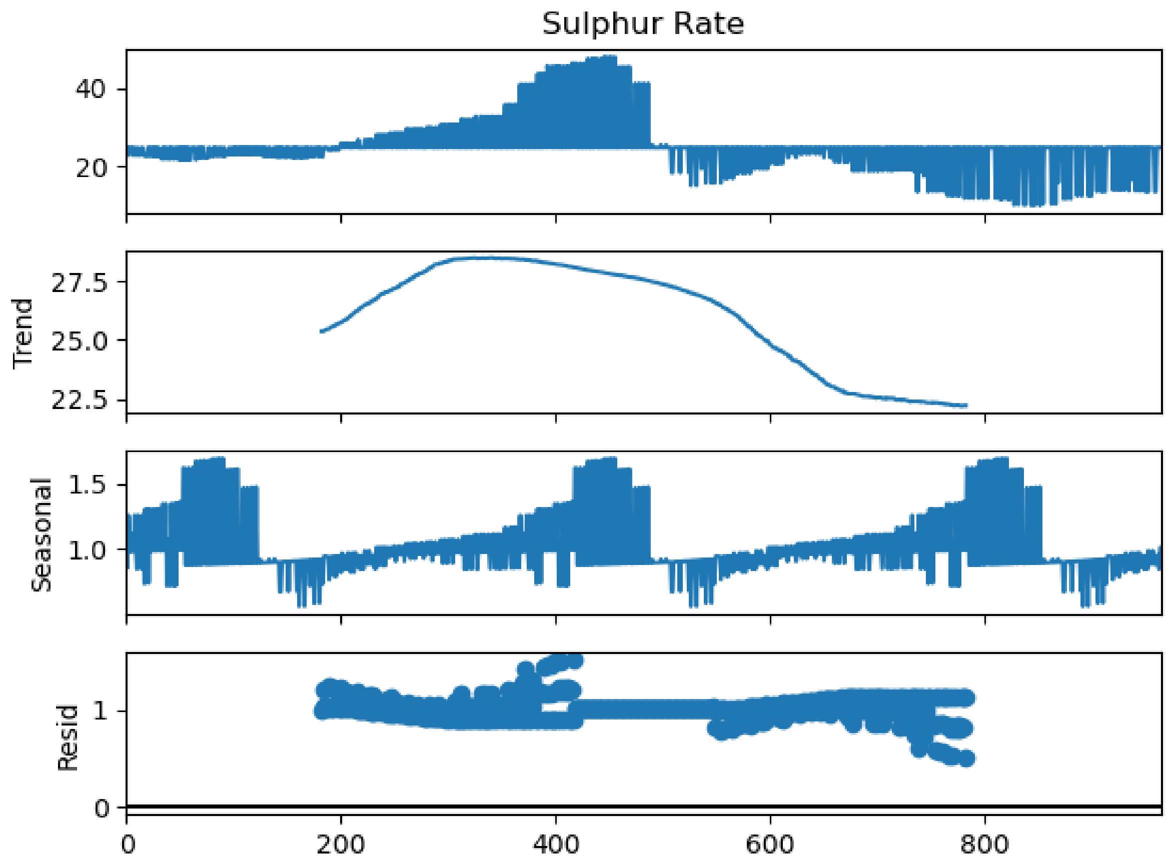
In [12]:

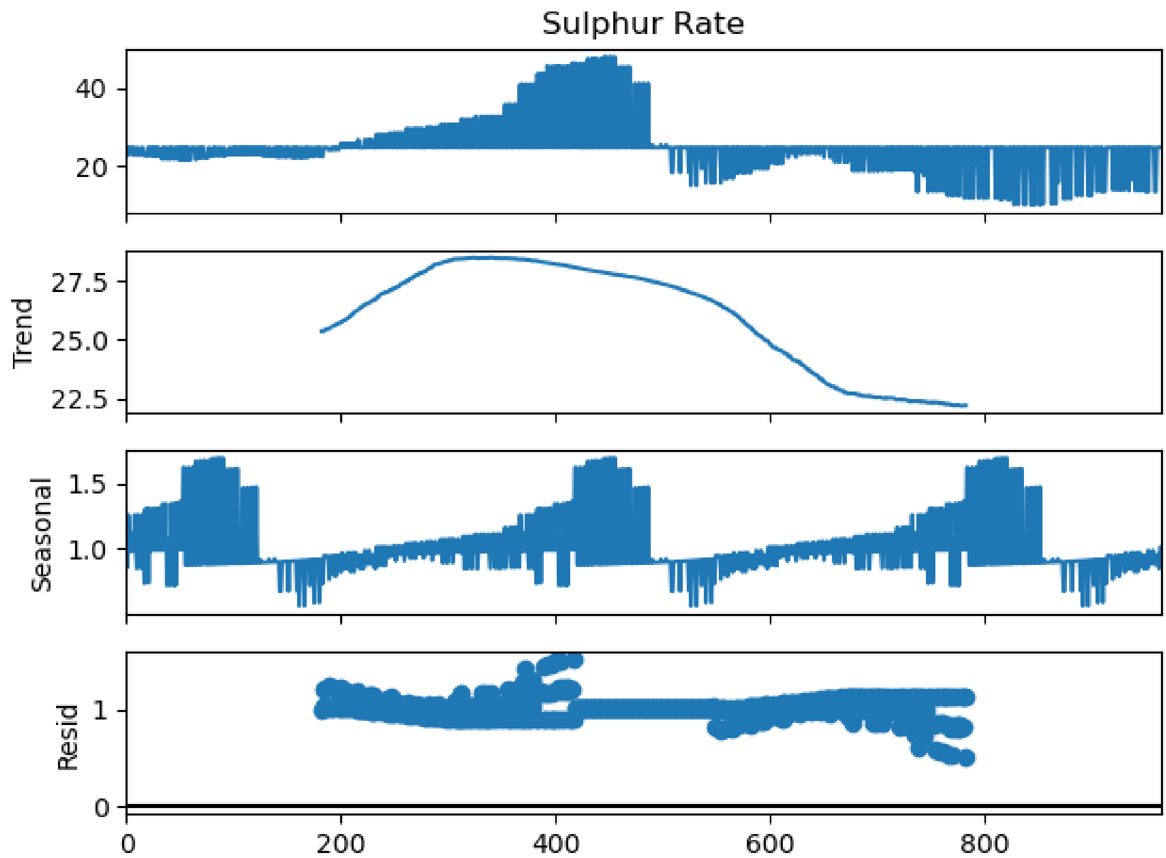
```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df['Sulphur Rate'],
                            model ='multiplicative', period = 365)
```

In [13]:

```
result.plot()
```

Out[13]:





ARIMA Model Preprocessing

In [14]:

```
#Train_test_split

spilt = (df.index < len(df)-30)
df_train = df[spilt].copy()
df_test = df[~spilt].copy()
```

In [15]:

```
#Stationarity checking

from statsmodels.tsa.stattools import adfuller

# Select the column 'Sulphur Rate' from the DataFrame
sulphur_rate_data = df_train['Sulphur Rate']

# Perform the Augmented Dickey-Fuller test
adf_test = adfuller(sulphur_rate_data)
print(f'p-value: {adf_test[1]}')    #This gives us a output of p-value: 0.6980113868350736
                                    #This result shows a large p-value, which means the tes
```

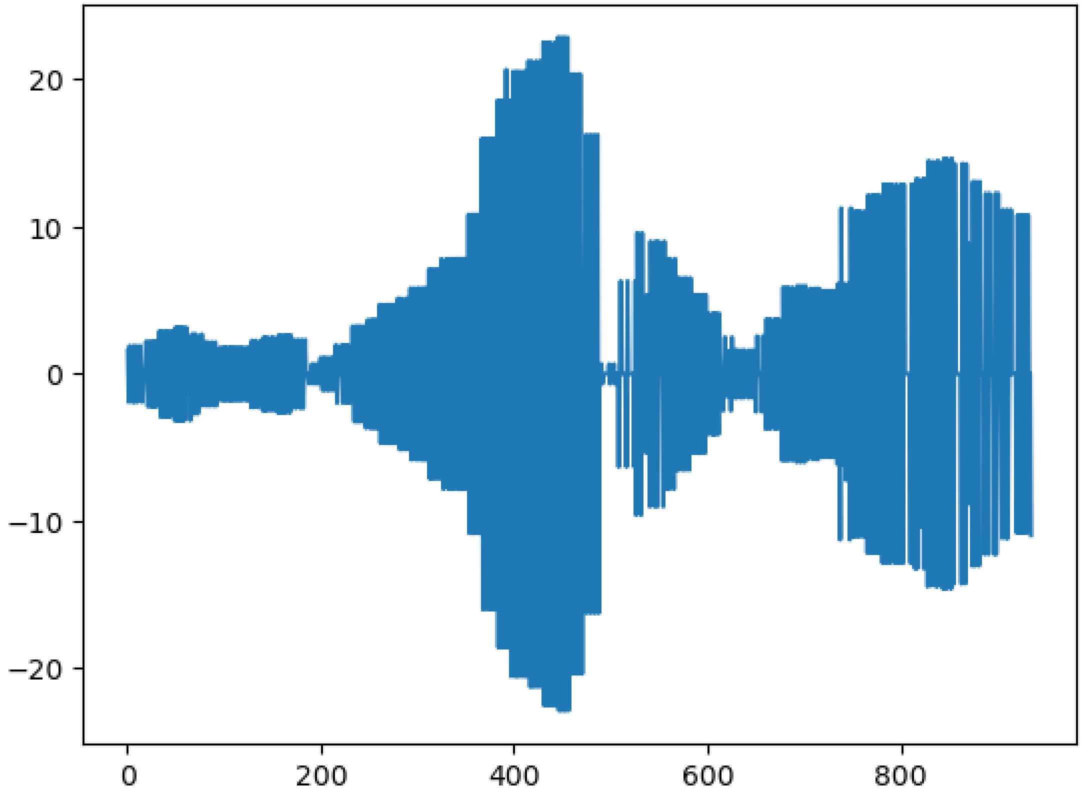
```
p-value: 0.6980113868350736
```

In [16]:

```
#First Differencing

df_train_diff = df_train['Sulphur Rate'].diff().dropna()
df_train_diff.plot()
```

Out[16]:

```
<Axes: >
```



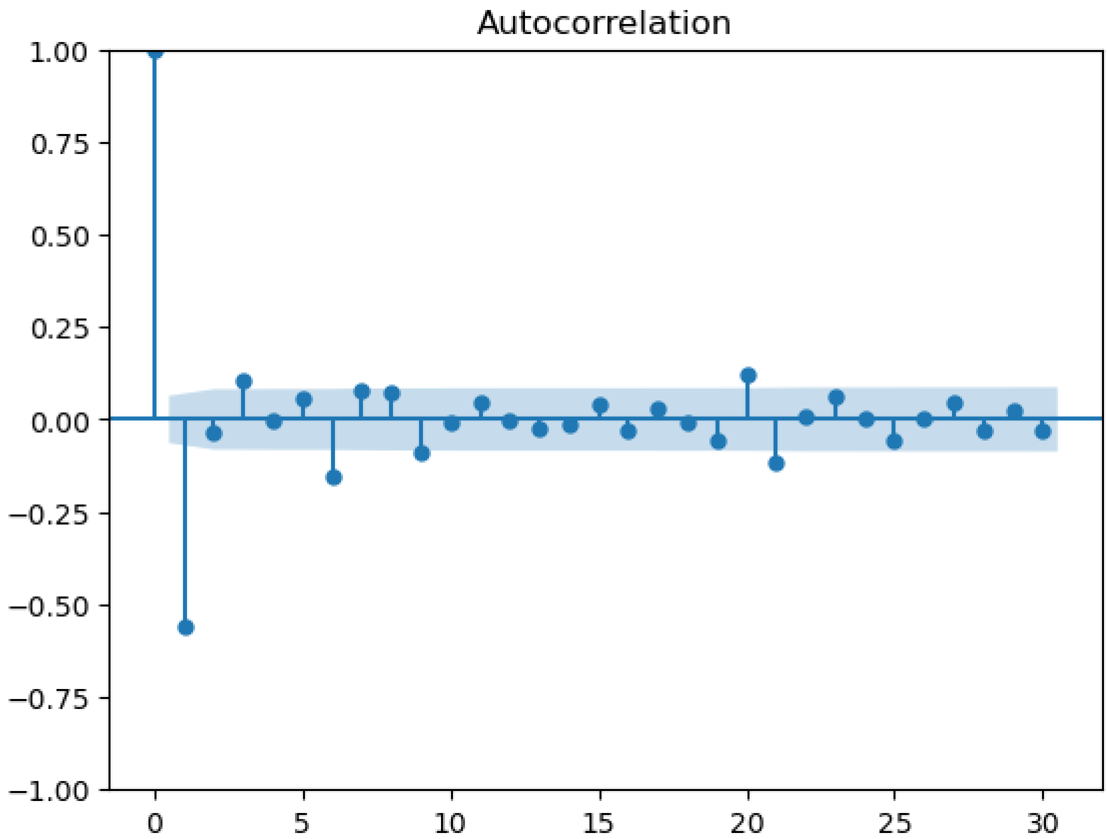
In [17]:

```
adf_test = adfuller(df_train_diff)
print(f'p-value: {adf_test[1]}') # This gives us a p-value:2.1229061985843574e-16  which is
```

```
p-value: 2.1229061985843574e-16
```

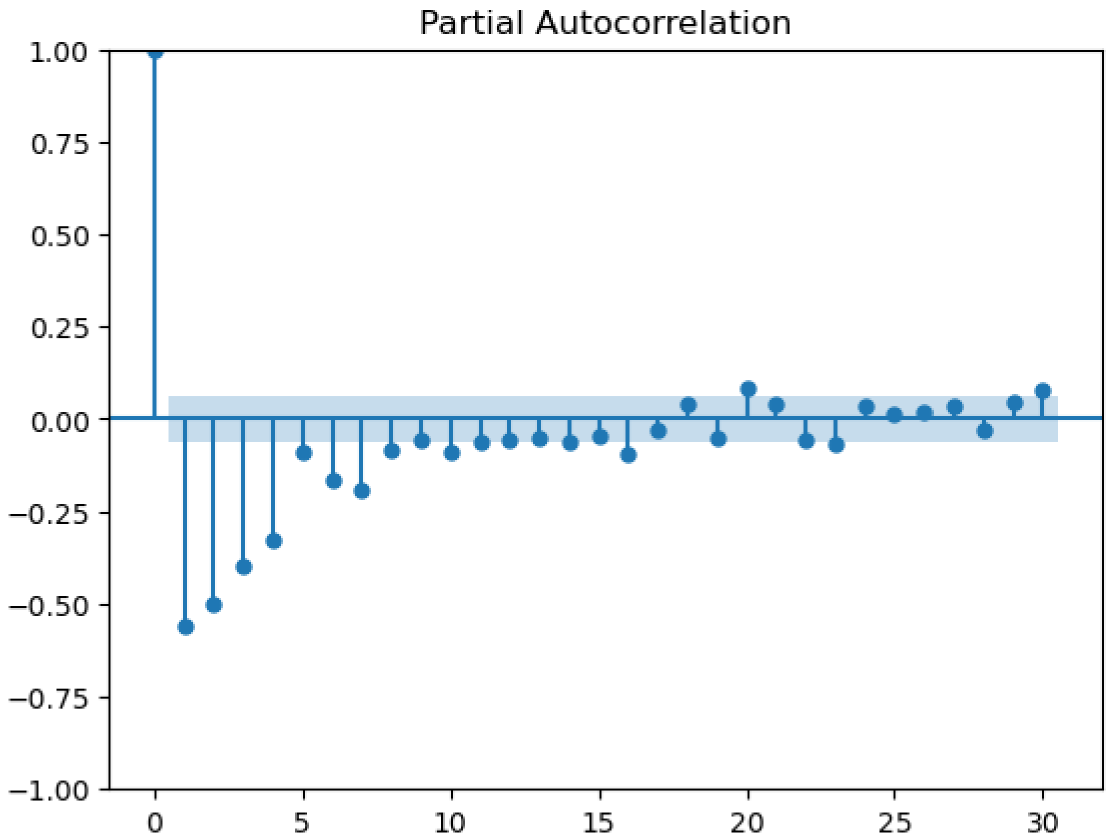
In [18]:

```
#Finding the parameters p and q

from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf
series = df_train_diff #insert data here
plot_acf(series) #ACF plot function
pyplot.show() #Show graph
```



In [19]:

```
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_pacf
series = df_train_diff #insert data here
plot_pacf(series) #ACF plot function
pyplot.show() #Show graph
```



Auto ARIMA

In [20]:

```
from pmdarima import auto_arima


model = auto_arima(df_train['Sulphur Rate'], seasonal=True, m=7, max_p=5, max_d=2, max_q=5,
print(model.summary())
```

```
                               SARIMAX Results                                
==============================================================================
==
Dep. Variable:                      y   No. Observations:                    9
35
Model:               SARIMAX(4, 1, 3)   Log Likelihood               -2715.8
12
Date:                Tue, 19 Mar 2024   AIC                           5447.6
24
Time:                        14:46:22   BIC                           5486.3
40
Sample:                             0   HQIC                          5462.3
87
                                - 935                                         
Covariance Type:                  opg                                         
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1         -1.3429      0.028    -48.262      0.000      -1.397      -1.2
88
ar.L2         -1.6313      0.040    -40.603      0.000      -1.710      -1.5
53
ar.L3         -0.7510      0.039    -19.398      0.000      -0.827      -0.6
75
ar.L4         -0.3408      0.024    -14.244      0.000      -0.388      -0.2
94
ma.L1          0.0379      0.020      1.860      0.063      -0.002       0.0
78
ma.L2          0.1339      0.020      6.607      0.000       0.094       0.1
74
ma.L3         -0.8279      0.021    -39.526      0.000      -0.869      -0.7
87
sigma2        19.5688      0.708     27.626      0.000      18.180      20.9
57
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.15   Jarque-Bera (JB):              
150.47
Prob(Q):                              0.70   Prob(JB):                      
0.00
Heteroskedasticity (H):               8.23   Skew:                          
-0.09
Prob(H) (two-sided):                  0.00   Kurtosis:                      
4.96
==============================================================================
=======
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

In [32]:

```
from sklearn.metrics import mean_squared_error, r2_score

n_forecast = len(df_test)  # Number of periods to forecast into the future
forecast = model.predict(n_periods=n_forecast)

# Step 9: Make Predictions
y_pred_test = forecast  # No need to call .values

test_score = r2_score(df_test['Sulphur Rate'], y_pred_test)
print("Test Score (R-squared)(ARIMA Model):", test_score)


# Step 10: Evaluate the Model
mse = mean_squared_error(df_test['Sulphur Rate'], y_pred_test)
rmse = np.sqrt(mse)
print("Root Mean Squared Error(ARIMA Model):", rmse)

# Print the forecasts
print("Forecasted values:")
print(y_pred_test)
```

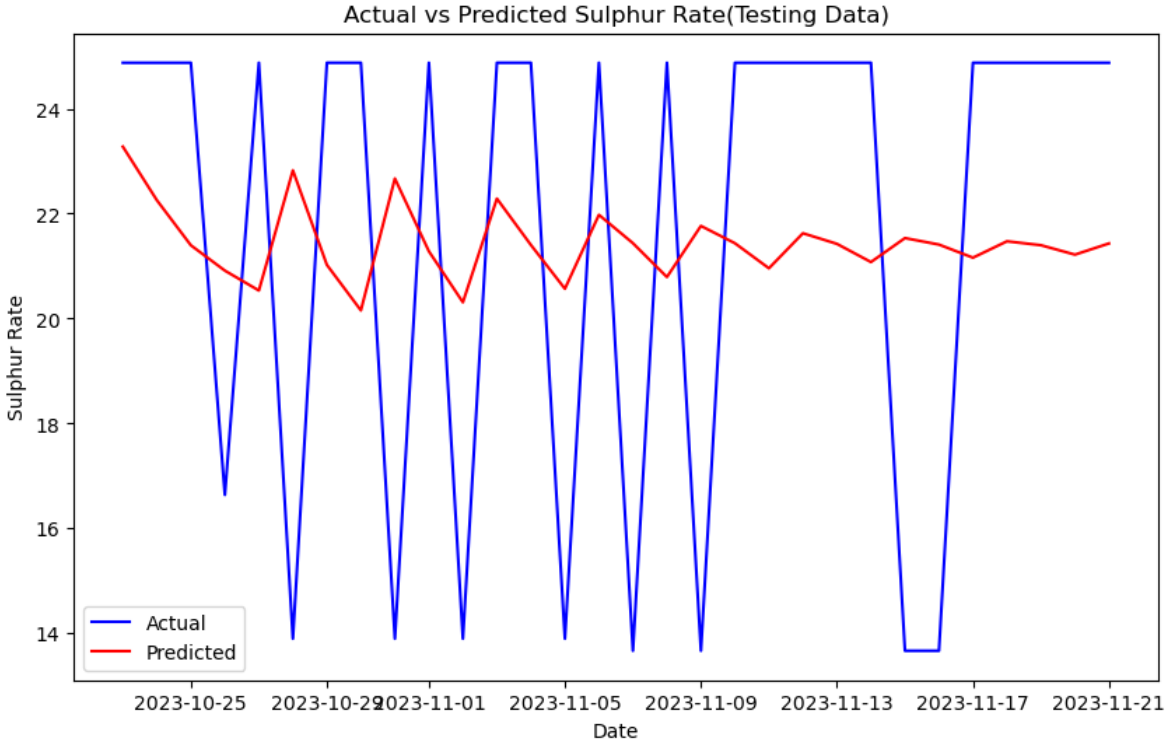
```
Test Score (R-squared)(ARIMA Model): -0.041210518480958 36
Root Mean Squared Error(ARIMA Model): 5.062778070696675
Forecasted values:
935    23.276484
936    22.255916
937    21.396848
938    20.914060
939    20.534919
940    22.824571
941    21.023602
942    20.156218
943    22.668735
944    21.281856
945    20.310689
946    22.286083
947    21.402895
948    20.568364
949    21.977316
950    21.436691
951    20.791923
952    21.766029
953    21.435574
954    20.958698
955    21.626375
956    21.423894
957    21.077341
958    21.534145
959    21.410567
960    21.160578
961    21.472937
962    21.398416
963    21.218780
964    21.432201
dtype: float64
```

In [30]:

```
plt.figure(figsize=(10, 6))
plt.plot(df_test['Date'], df_test['Sulphur Rate'], label='Actual', color='blue')
plt.plot(df_test['Date'], y_pred_test, label='Predicted', color='red')
plt.title('Actual vs Predicted Sulphur Rate(Testing Data)')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
plt.legend()
```

Out[30]:

```
<matplotlib.legend.Legend at 0xcae80c15d0>
```

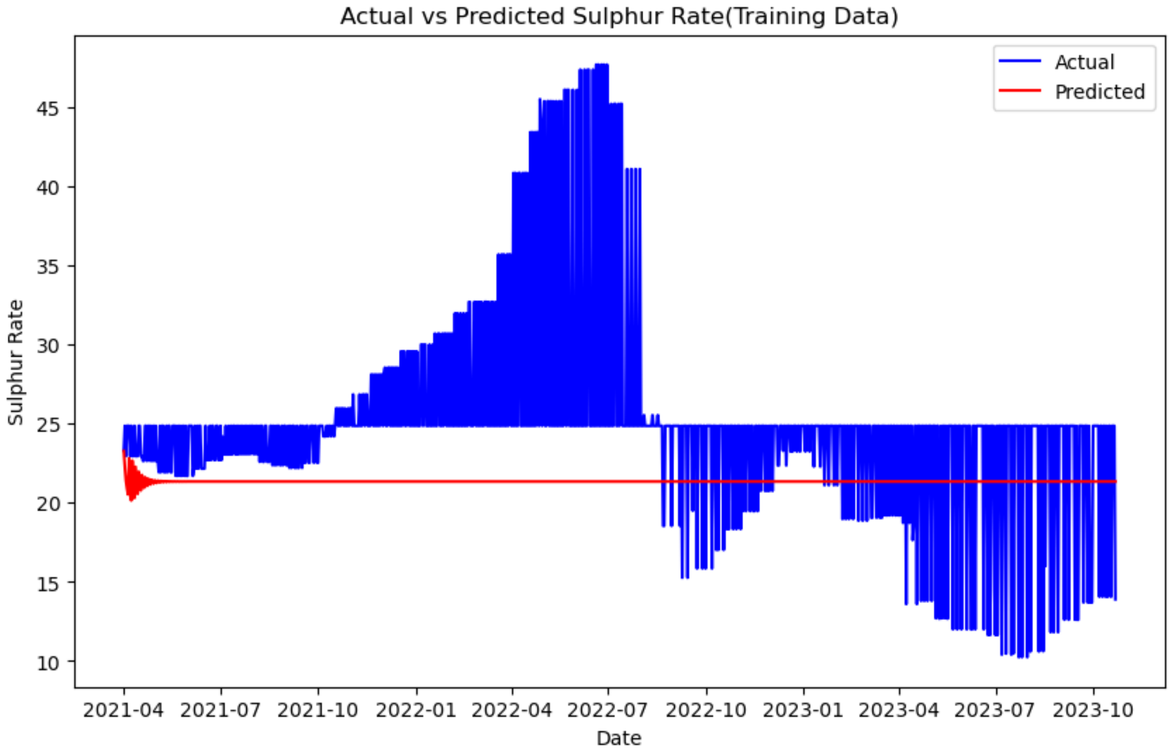


In [29]:

```
forecast = model.predict(n_periods=len(df_train))
y_pred_train = forecast


plt.figure(figsize=(10, 6))
plt.plot(df_train['Date'], df_train['Sulphur Rate'], label='Actual', color='blue')
plt.plot(df_train['Date'], y_pred_train, label='Predicted', color='red')
plt.title('Actual vs Predicted Sulphur Rate(Training Data)')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
plt.legend()
```

Out[29]:

```
<matplotlib.legend.Legend at 0xcae8345c50>
```



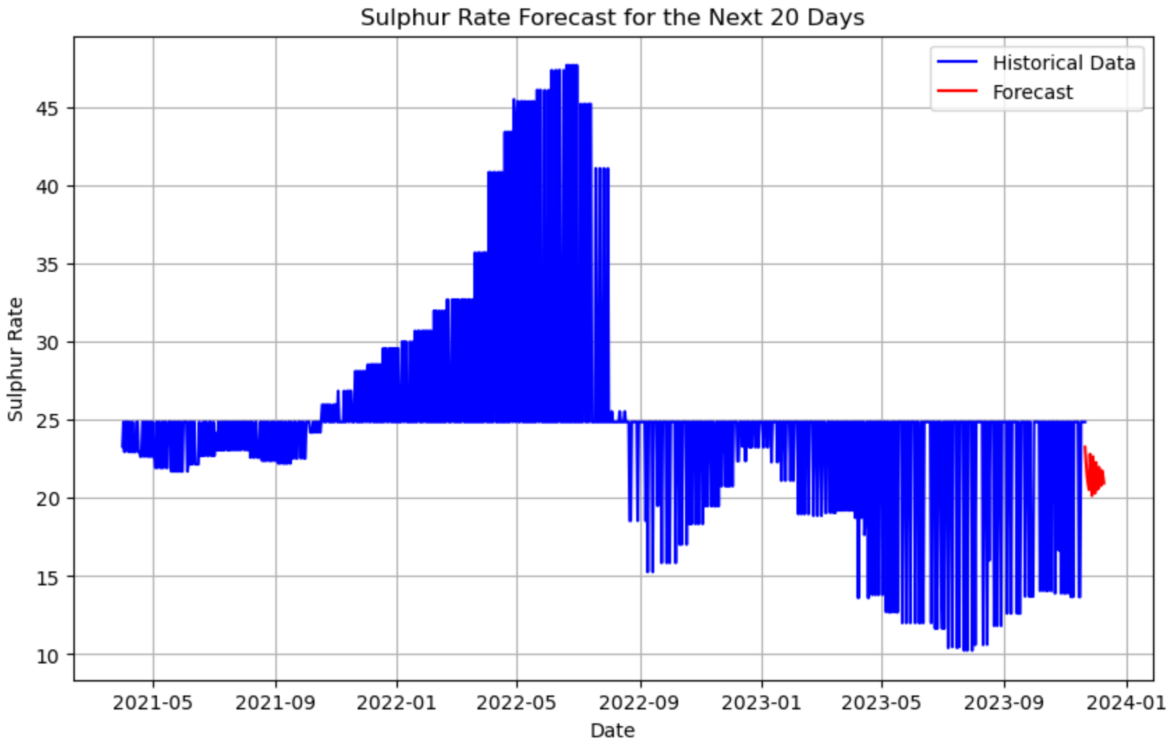
In [26]:

```
#Generate forecasts for the next 20 days
forecast_next_20_days = model.predict(n_periods=20)

# Print the forecasted values
print("Forecast for the next 20 days:")
print(forecast_next_20_days)
```

```
Forecast for the next 20 days:
935    23.276484
936    22.255916
937    21.396848
938    20.914060
939    20.534919
940    22.824571
941    21.023602
942    20.156218
943    22.668735
944    21.281856
945    20.310689
946    22.286083
947    21.402895
948    20.568364
949    21.977316
950    21.436691
951    20.791923
952    21.766029
953    21.435574
954    20.958698
dtype: float64
```

In [28]:

```
# Plot the forecasted values along with the existing data
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Sulphur Rate'], label='Historical Data', color='blue')
plt.plot(pd.date_range(df['Date'].iloc[-1], periods=20), forecast_next_20_days, label='Fore
plt.title('Sulphur Rate Forecast for the Next 20 Days')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
plt.legend()
plt.grid(True)
plt.show()
```



Manual ARIMA

In [34]:

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train['Sulphur Rate'], order=(2,1,1))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
==
Dep. Variable:           Sulphur Rate   No. Observations:                    9
35
Model:                 ARIMA(2, 1, 1)   Log Likelihood               -2736.4
87
Date:                Tue, 19 Mar 2024   AIC                           5480.9
74
Time:                        16:46:08   BIC                           5500.3
31
Sample:                             0   HQIC                          5488.3
55
                                - 935
Covariance Type:                  opg
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1         -0.4170      0.023    -17.760      0.000      -0.463      -0.3
71
ar.L2         -0.2955      0.022    -13.615      0.000      -0.338      -0.2
53
ma.L1         -0.8884      0.012    -76.439      0.000      -0.911      -0.8
66
sigma2        20.4676      0.751     27.258      0.000      18.996      21.9
39
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.75   Jarque-Bera (JB):
130.96
Prob(Q):                              0.39   Prob(JB):
0.00
Heteroskedasticity (H):               9.17   Skew:
-0.04
Prob(H) (two-sided):                  0.00   Kurtosis:
4.83
==============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

In [61]:

```
from sklearn.metrics import mean_squared_error, r2_score

# Step 8: Validate the Model
# Forecast
forecast = model_fit.forecast(steps=len(df_test))

# Step 9: Make Predictions
predictions = forecast.values

# Step 10: Evaluate the Model
mse = mean_squared_error(df_test['Sulphur Rate'], predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 4.929557193347942
```

In [62]:

```
forecast = model_fit.forecast(steps=len(df_train))
predictions = forecast.values

y_true = df_train['Sulphur Rate'].values
y_pred = forecast.values

# Calculate R-squared score
test_score = r2_score(y_true, y_pred)
print("Test Score (R-squared):", test_score)

print("Predicted values:",y_pred)
print("True values:",y_true)
```

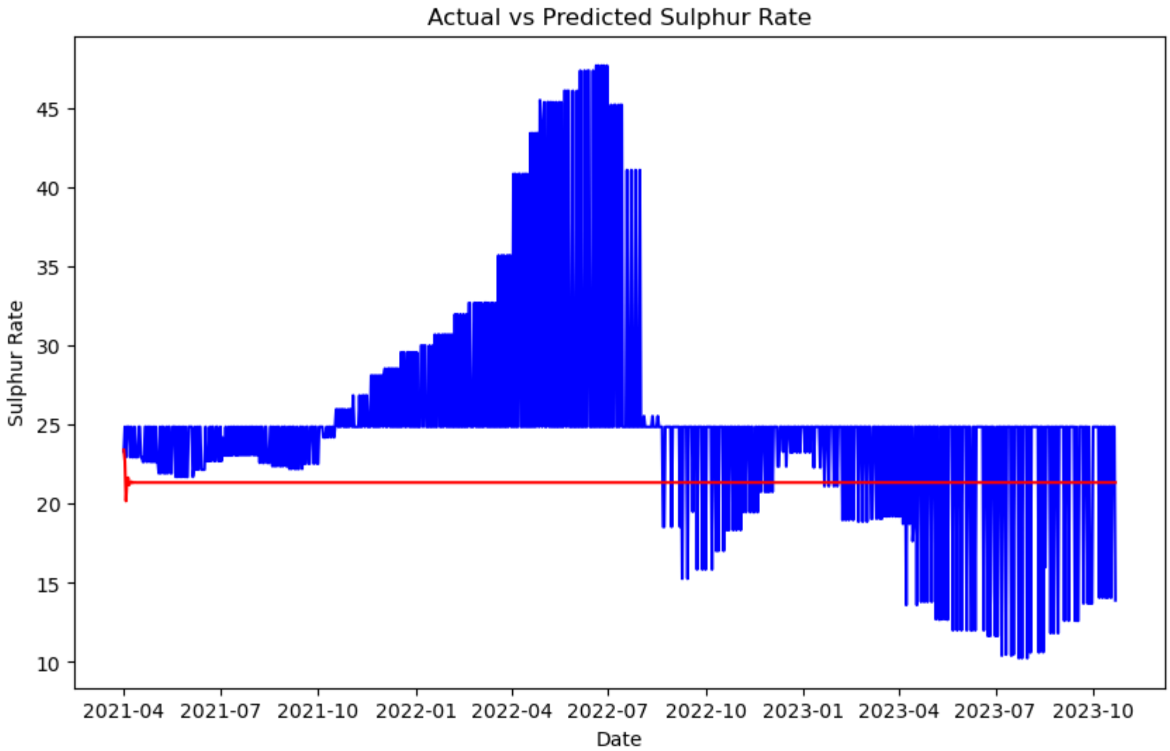
```
Test Score (R-squared): -0.3680812793680073
Predicted values: [23.41976753 22.69383408 20.18311815 21.4445569  21.6605
7163 21.19770183
 21.3268665  21.40980186 21.33704755 21.34287396 21.36194584 21.35227148
 21.35066905 21.35419632 21.35319911 21.35257249 21.35312848 21.35308184
 21.35293697 21.35301116 21.35302304 21.35299616 21.35300386 21.35300859
 21.35300434 21.35300472 21.35300582 21.35300525 21.35300516 21.35300537
 21.35300531 21.35300527 21.3530053  21.3530053  21.35300529 21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
 21.3530053  21.3530053  21.3530053  21.3530053  21.3530053  21.3530053
```

In [63]:

```
plt.figure(figsize=(10, 6))
plt.plot(df_train['Date'], df_train['Sulphur Rate'], label='Actual', color='blue')
plt.plot(df_train['Date'], predictions, label='Predicted', color='red')
plt.title('Actual vs Predicted Sulphur Rate')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
```

Out[63]:

```
Text(0, 0.5, 'Sulphur Rate')
```



In [64]:

```
from sklearn.metrics import r2_score

forecast = model_fit.forecast(steps=len(df_test))
predictions = forecast.values

# Assuming 'Sulphur Rate' is the target variable
y_true = df_test['Sulphur Rate'].values
y_pred = forecast.values

# Calculate R-squared score
test_score = r2_score(y_true, y_pred)
print("Test Score (R-squared):", test_score)

print("Predicted values:",y_pred)
print("True values:",y_true)
```

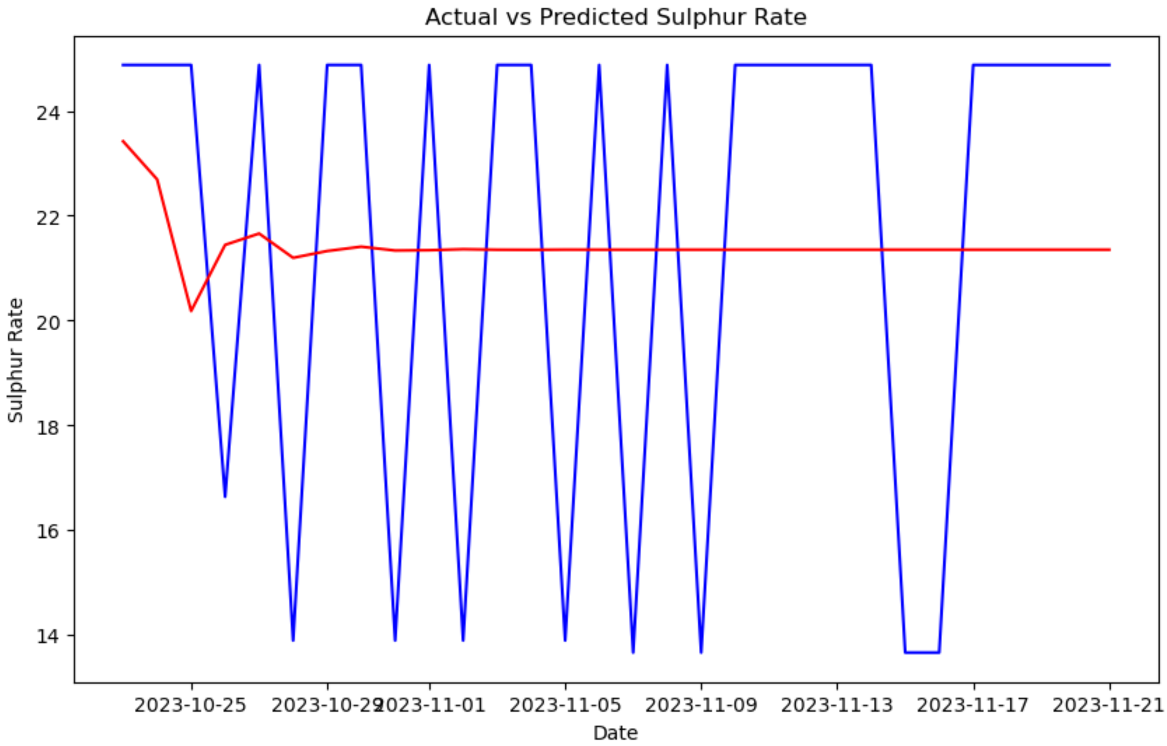
```
Test Score (R-squared): 0.012864920380618261
Predicted values: [23.41976753 22.69383408 20.18311815 21.4445569  21.660571
63 21.19770183
 21.3268665  21.40980186 21.33704755 21.34287396 21.36194584 21.35227148
 21.35066905 21.35419632 21.35319911 21.35257249 21.35312848 21.35308184
 21.35293697 21.35301116 21.35302304 21.35299616 21.35300386 21.35300859
 21.35300434 21.35300472 21.35300582 21.35300525 21.35300516 21.35300537]
True values: [24.87524725 24.87524725 24.87524725 16.64       24.87524725 1
3.9
 24.87524725 24.87524725 13.9        24.87524725 13.9        24.87524725
 24.87524725 13.9        24.87524725 13.67       24.87524725 13.67
 24.87524725 24.87524725 24.87524725 24.87524725 24.87524725 13.67
 13.67       24.87524725 24.87524725 24.87524725 24.87524725 24.87524725]
```

In [65]:

```
plt.figure(figsize=(10, 6))
plt.plot(df_test['Date'], df_test['Sulphur Rate'], label='Actual', color='blue')
plt.plot(df_test['Date'], predictions, label='Predicted', color='red')
plt.title('Actual vs Predicted Sulphur Rate')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
```

Out[65]:

```
Text(0, 0.5, 'Sulphur Rate')
```



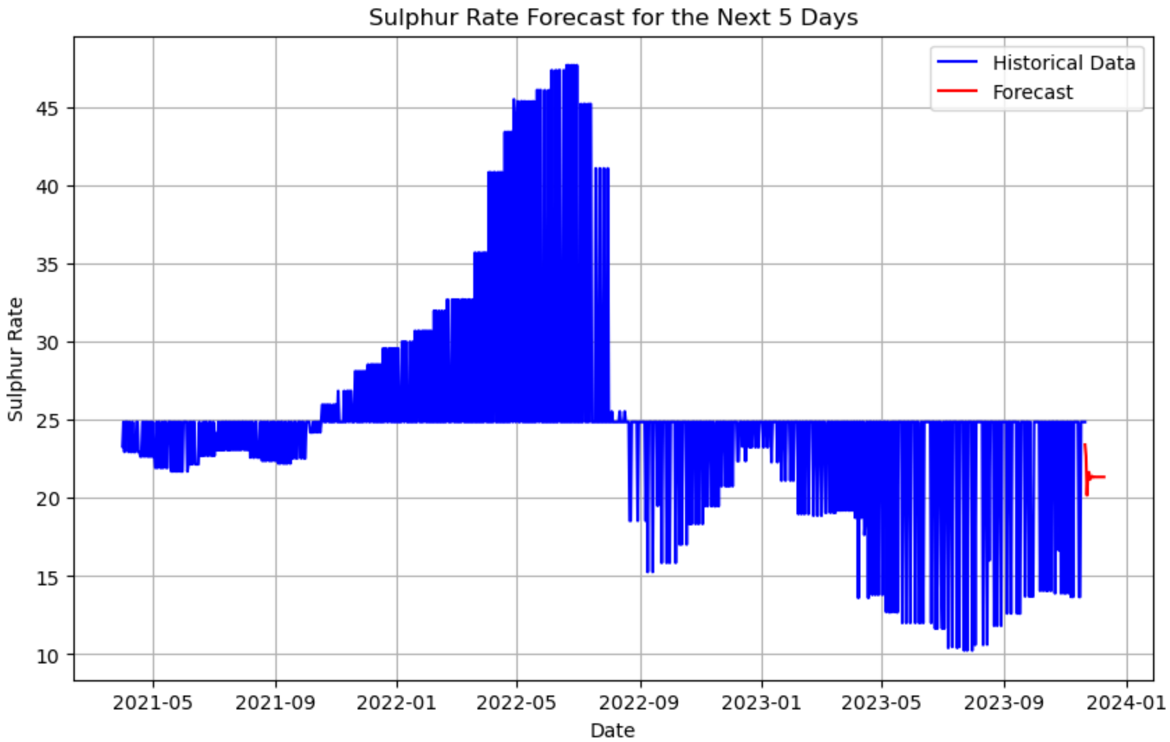
In [66]:

```
forecast_next_20_days = model_fit.forecast(steps=20)

# Print the forecasted values
print("Forecast for the next 20 days:")
print(forecast_next_20_days)
```

```
Forecast for the next 20 days:
935    23.419768
936    22.693834
937    20.183118
938    21.444557
939    21.660572
940    21.197702
941    21.326866
942    21.409802
943    21.337048
944    21.342874
945    21.361946
946    21.352271
947    21.350669
948    21.354196
949    21.353199
950    21.352572
951    21.353128
952    21.353082
953    21.352937
954    21.353011
Name: predicted_mean, dtype: float64
```

In [67]:

```
# Plot the forecasted values along with the existing data
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Sulphur Rate'], label='Historical Data', color='blue')
plt.plot(pd.date_range(df['Date'].iloc[-1], periods=20), forecast_next_20_days, label='Fore
plt.title('Sulphur Rate Forecast for the Next 5 Days')
plt.xlabel('Date')
plt.ylabel('Sulphur Rate')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

```

```

VAR (Vector Autoregression Model)

In [38]:

```
df1 = pd.read_excel(r'sulphur and sulphuric acid daily data.xlsx')
```

In [39]:

```
df1
```

Out[39]:

	Date	Sulphur Rate	Sulphuric acid Rate
0	2021-04-01	23.31	NaN
1	2021-04-02	NaN	10.07
2	2021-04-03	22.96	10.07
3	2021-04-04	NaN	10.07
4	2021-04-05	NaN	10.07
...	...	...	...
960	2023-11-17	NaN	NaN
961	2023-11-18	NaN	NaN
962	2023-11-19	NaN	NaN
963	2023-11-20	NaN	NaN
964	2023-11-21	NaN	NaN

965 rows × 3 columns

In [40]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 965 entries, 0 to 964
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0    Date                965 non-null    datetime64[ns]
 1   Sulphur Rate         364 non-null    float64
 2   Sulphuric acid Rate  398 non-null    float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 22.7 KB
```

In [41]:

```
from sklearn.impute import KNNImputer

# Initialize the KNN imputer
k = 5  # Number of neighbors to consider
imputer = KNNImputer(n_neighbors=k)

df_subset = df1[['Sulphur Rate']]
df_imputed_sulphur = imputer.fit_transform(df_subset)
df1['Sulphur Rate'] = df_imputed_sulphur
```

In [42]:

```
df1
```

Out[42]:

	Date	Sulphur Rate	Sulphuric acid Rate
0	2021-04-01	23.310000	NaN
1	2021-04-02	24.875247	10.07
2	2021-04-03	22.960000	10.07
3	2021-04-04	24.875247	10.07
4	2021-04-05	24.875247	10.07
...	...	...	...
960	2023-11-17	24.875247	NaN
961	2023-11-18	24.875247	NaN
962	2023-11-19	24.875247	NaN
963	2023-11-20	24.875247	NaN
964	2023-11-21	24.875247	NaN

965 rows × 3 columns

In [43]:

```
# Initialize the KNN imputer
k = 30  # Number of neighbors to consider
imputer = KNNImputer(n_neighbors=k)

df_subset = df1[['Sulphuric acid Rate']]
df_imputed_sulphuric  = imputer.fit_transform(df_subset)
df1['Sulphuric acid Rate'] = df_imputed_sulphuric
```

In [44]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 965 entries, 0 to 964
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype         
---  ------               --------------  -----         
 0    Date                965 non-null    datetime64[ns]
 1   Sulphur Rate         965 non-null    float64       
 2   Sulphuric acid Rate  965 non-null    float64       
dtypes: datetime64[ns](1), float64(2)
memory usage: 22.7 KB
```

In [45]:

```
#Stationarity checking

from statsmodels.tsa.stattools import adfuller

# Select the column 'Sulphur Rate' from the DataFrame
sulphur_rate_data = df1['Sulphur Rate']

# Perform the Augmented Dickey-Fuller test
adf_test = adfuller(sulphur_rate_data)
print(f'p-value: {adf_test[1]}')    #This gives us a output of p-value: 0.6980113868350736
                                    #This result shows a large p-value, which means the tes
```

```
p-value: 0.682887920692923
```

In [46]:

```
#Stationarity checking

from statsmodels.tsa.stattools import adfuller

# Select the column 'Sulphur acid Rate' from the DataFrame
sulphur_rate_data = df1['Sulphuric acid Rate']

# Perform the Augmented Dickey-Fuller test
adf_test = adfuller(sulphur_rate_data)
print(f'p-value: {adf_test[1]}')    #This gives us a output of p-value: 0.6980113868350736
                                    #This result shows a large p-value, which means the tes
```

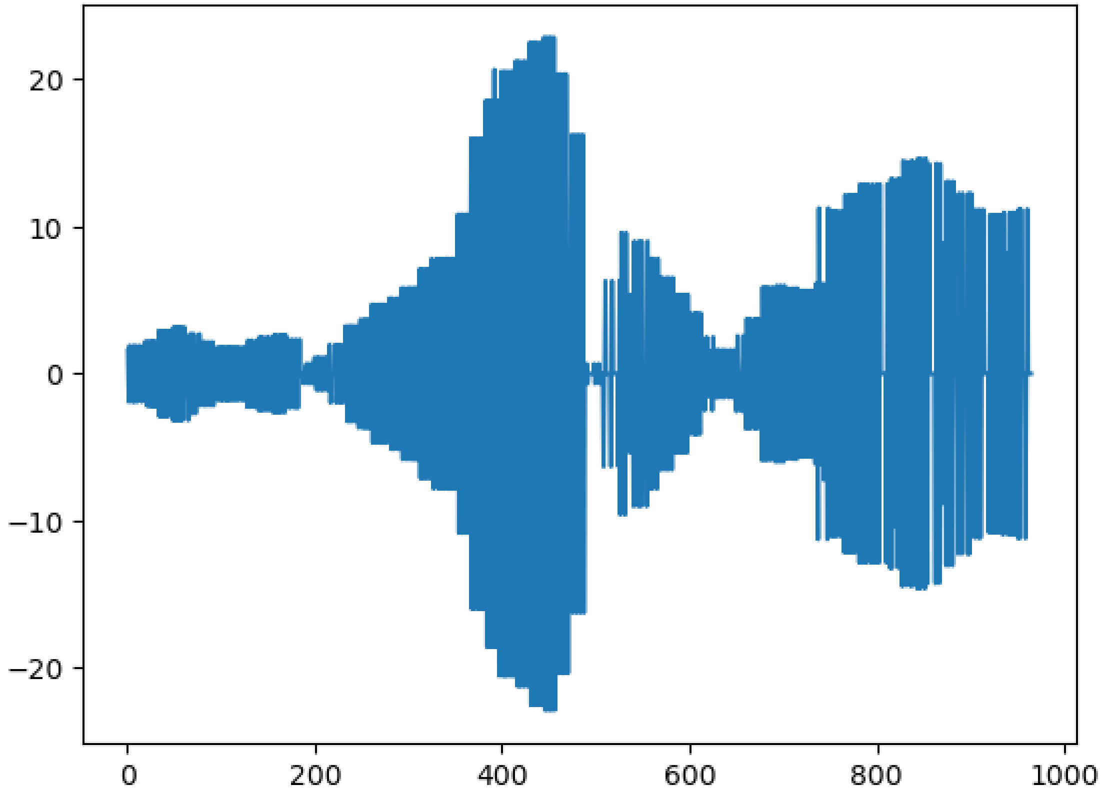
```
p-value: 0.0033519626650673554
```

In [47]:

```
#First Differencing

df_diff = df1['Sulphur Rate'].diff().dropna()
df_diff.plot()
```

Out[47]:

```
<Axes: >
```



In [51]:

```
#Stationarity checking

from statsmodels.tsa.stattools import adfuller

# Select the column 'Sulphur Rate' from the DataFrame
sulphur_rate_data = df_diff

# Perform the Augmented Dickey-Fuller test
adf_test = adfuller(sulphur_rate_data)
print(f'p-value: {adf_test[1]}')    #This gives us a output of p-value: 0.6980113868350736
                                    #This result shows a large p-value, which means the tes
```

```
p-value: 2.123006347762479e-17
```

In [54]:

```
from sklearn.model_selection import train_test_split
from statsmodels.tsa.api import VAR

# Step 1: Check Stationarity
acid_rate_stationary = adfuller(df1['Sulphuric acid Rate'])[1] < 0.05

# Step 2: Make "Sulphur Rate" Data Stationary
if not acid_rate_stationary:
    df['Sulphur Rate Diff'] = df1['Sulphur Rate'].diff().dropna()

# Step 3: Split the Data into Training and Testing Sets
train_df, test_df = train_test_split(df1, test_size=0.2, shuffle=False)  # Adjust test_size

# Step 4: Fit VAR Model on the Training Set
if acid_rate_stationary:
    model = VAR(train_df[['Sulphur Rate', 'Sulphuric acid Rate']])
else:
    model = VAR(train_df[['Sulphur Rate Diff', 'Sulphuric acid Rate']])

results = model.fit()
```

In [55]:

```
results.summary()
```

Out[55]:

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Tue, 19, Mar, 2024
Time:                     16:54:19
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                    4.86708
Nobs:                     771.000    HQIC:                   4.84483
Log likelihood:          -4044.32    FPE:                    125.326
AIC:                      4.83092    Det(Omega_mle):         124.356
--------------------------------------------------------------------
Results for equation Sulphur Rate
============================================================================
=============
                            coefficient       std. error           t-stat
prob
----------------------------------------------------------------------------
-------------
const                         20.916100         1.385496           15.096
0.000
L1.Sulphur Rate                0.175403         0.035664            4.918
0.000
L1.Sulphuric acid Rate         0.030350         0.098566            0.308
0.758
============================================================================
=============

Results for equation Sulphuric acid Rate
============================================================================
=============
                            coefficient       std. error           t-stat
prob
----------------------------------------------------------------------------
-------------
const                          6.279709         0.478278           13.130
0.000
L1.Sulphur Rate                0.015249         0.012311            1.239
0.215
L1.Sulphuric acid Rate         0.333257         0.034025            9.794
0.000
============================================================================
=============

Correlation matrix of residuals
                       Sulphur Rate  Sulphuric acid Rate
Sulphur Rate               1.000000            -0.046015
Sulphuric acid Rate       -0.046015             1.000000
```

In [ ]: