A Statistical study of weather prediction

In [1]:
```
# import the libraries
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from scipy.stats import zscore
from sklearn.preprocessing import RobustScaler
```

In [2]:
```
# Load the dataset
df = pd.read_excel('W Data.xlsx')
df
```

Out[2]:

	country	location_name	timezone	last_updated	temperature_celsius	condition_text	wind_kph	wind_degree	wind_direction	pressure_in	humidit
0	India	New Delhi	Asia/Nicosia	2023-08-29 15:00:00	34.0	Mist	6.8	250	WSW	29.65	5
1	India	New Delhi	Asia/Nicosia	2023-08-30 08:30:00	29.0	Mist	11.2	260	W	29.74	6
2	India	New Delhi	Asia/Nicosia	2023-08-31 05:15:00	29.0	Mist	3.6	211	SSW	29.74	7
3	India	New Delhi	Asia/Nicosia	2023-09-01 05:15:00	29.0	Mist	6.8	270	W	29.71	7
4	India	New Delhi	Asia/Nicosia	2023-09-02 05:00:00	31.3	Clear	12.6	274	W	29.64	3
...	...	...	...	...	...	...	...	...	...	...	.
91	India	New Delhi	Asia/Nicosia	2023-12-02 00:45:00	20.8	Partly cloudy	6.1	348	NNW	29.99	4
92	India	New Delhi	Asia/Nicosia	2023-12-04 02:00:00	18.0	Mist	3.6	10	N	30.03	8
93	India	New Delhi	Asia/Nicosia	2023-12-06 01:15:00	14.0	Mist	3.6	10	N	30.03	8
94	India	New Delhi	Asia/Nicosia	2023-12-07 01:15:00	16.0	Mist	3.6	10	N	30.03	7
95	India	New Delhi	Asia/Nicosia	2023-12-08 01:00:00	16.0	Mist	3.6	10	N	30.00	8

96 rows × 14 columns

In [3]:

```
# it give information about the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   country              96 non-null     object
 1   location_name        96 non-null     object
 2   timezone             96 non-null     object
 3   last_updated         96 non-null     datetime64[ns]
 4   temperature_celsius  96 non-null     float64
 5   condition_text       96 non-null     object
 6   wind_kph             96 non-null     float64
 7   wind_degree          96 non-null     int64
 8   wind_direction       96 non-null     object
 9   pressure_in          96 non-null     float64
 10  humidity             96 non-null     int64
 11  cloud                96 non-null     int64
 12  feels_like_celsius   96 non-null     float64
 13  visibility_km        96 non-null     float64
dtypes: datetime64[ns](1), float64(5), int64(3), object(5)
memory usage: 10.6+ KB
```

In [4]:

```
# check the missing values
df.isnull().sum()
```

Out[4]:

```
country                0
location_name          0
timezone               0
last_updated           0
temperature_celsius    0
condition_text         0
wind_kph               0
wind_degree            0
wind_direction         0
pressure_in            0
humidity               0
cloud                  0
feels_like_celsius     0
visibility_km          0
dtype: int64
```

In [5]:

```
# check duplicated values
df[df.duplicated()]
```

Out[5]:

country	location_name	timezone	last_updated	temperature_celsius	condition_text	wind_kph	wind_degree	wind_direction	pressure_in	humidity	cl

In [6]:

```
# drop null values
df.dropna()
```

Out[6]:

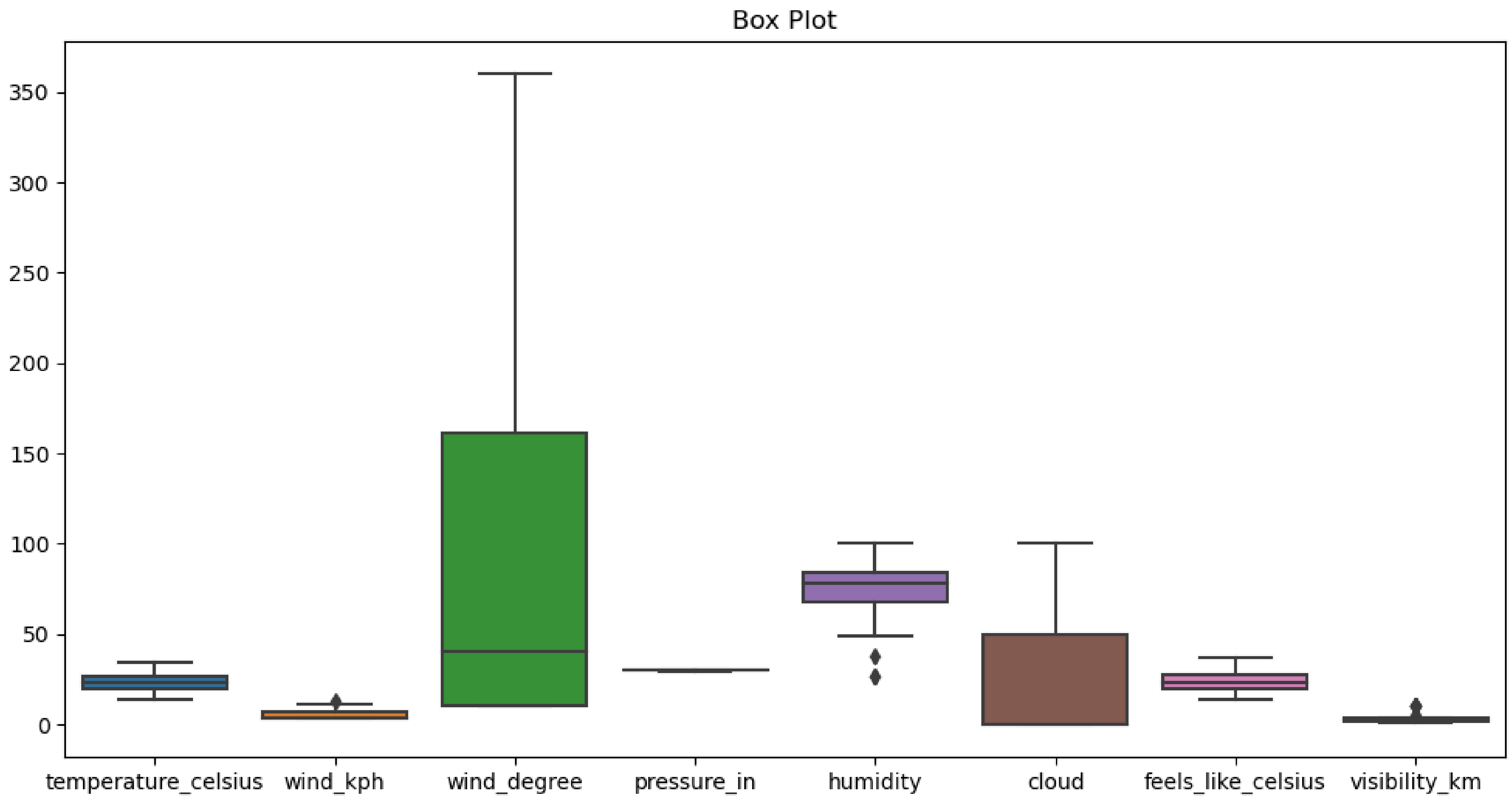
	country	location_name	timezone	last_updated	temperature_celsius	condition_text	wind_kph	wind_degree	wind_direction	pressure_in	humidit
0	India	New Delhi	Asia/Nicosia	2023-08-29 15:00:00	34.0	Mist	6.8	250	WSW	29.65	5
1	India	New Delhi	Asia/Nicosia	2023-08-30 08:30:00	29.0	Mist	11.2	260	W	29.74	6
2	India	New Delhi	Asia/Nicosia	2023-08-31 05:15:00	29.0	Mist	3.6	211	SSW	29.74	7
3	India	New Delhi	Asia/Nicosia	2023-09-01 05:15:00	29.0	Mist	6.8	270	W	29.71	7
4	India	New Delhi	Asia/Nicosia	2023-09-02 05:00:00	31.3	Clear	12.6	274	W	29.64	3
...	...	...	...	...	...	...	...	...	...	...	.
91	India	New Delhi	Asia/Nicosia	2023-12-02 00:45:00	20.8	Partly cloudy	6.1	348	NNW	29.99	4
92	India	New Delhi	Asia/Nicosia	2023-12-04 02:00:00	18.0	Mist	3.6	10	N	30.03	8
93	India	New Delhi	Asia/Nicosia	2023-12-06 01:15:00	14.0	Mist	3.6	10	N	30.03	8
94	India	New Delhi	Asia/Nicosia	2023-12-07 01:15:00	16.0	Mist	3.6	10	N	30.03	7
95	India	New Delhi	Asia/Nicosia	2023-12-08 01:00:00	16.0	Mist	3.6	10	N	30.00	8

96 rows × 14 columns

Exploratory Data Analysis

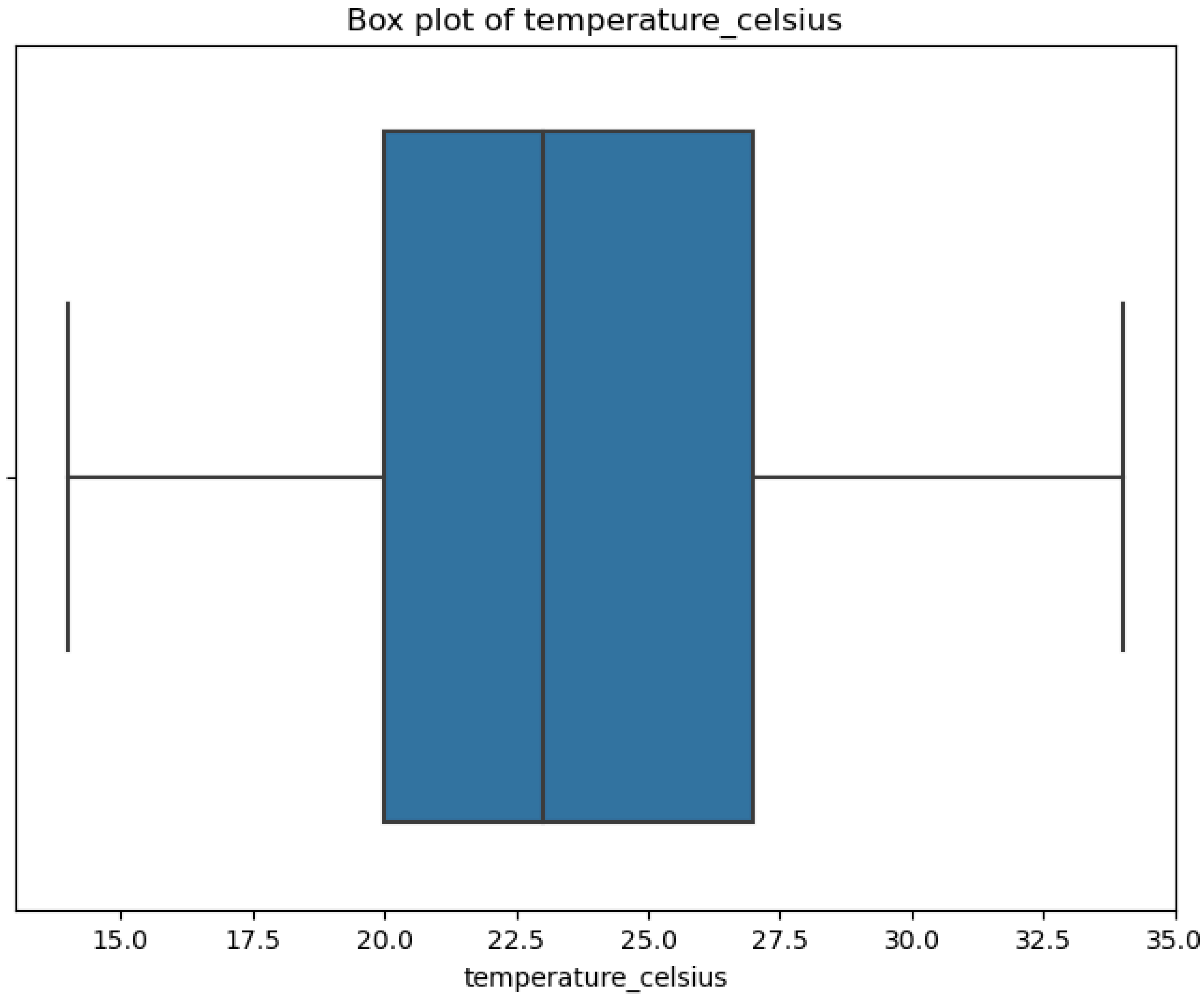
In [7]:
```
# only numerical columns
numerical_columns = ['temperature_celsius', 'wind_kph', 'wind_degree', 'pressure_in',
                     'humidity', 'cloud', 'feels_like_celsius', 'visibility_km']
```

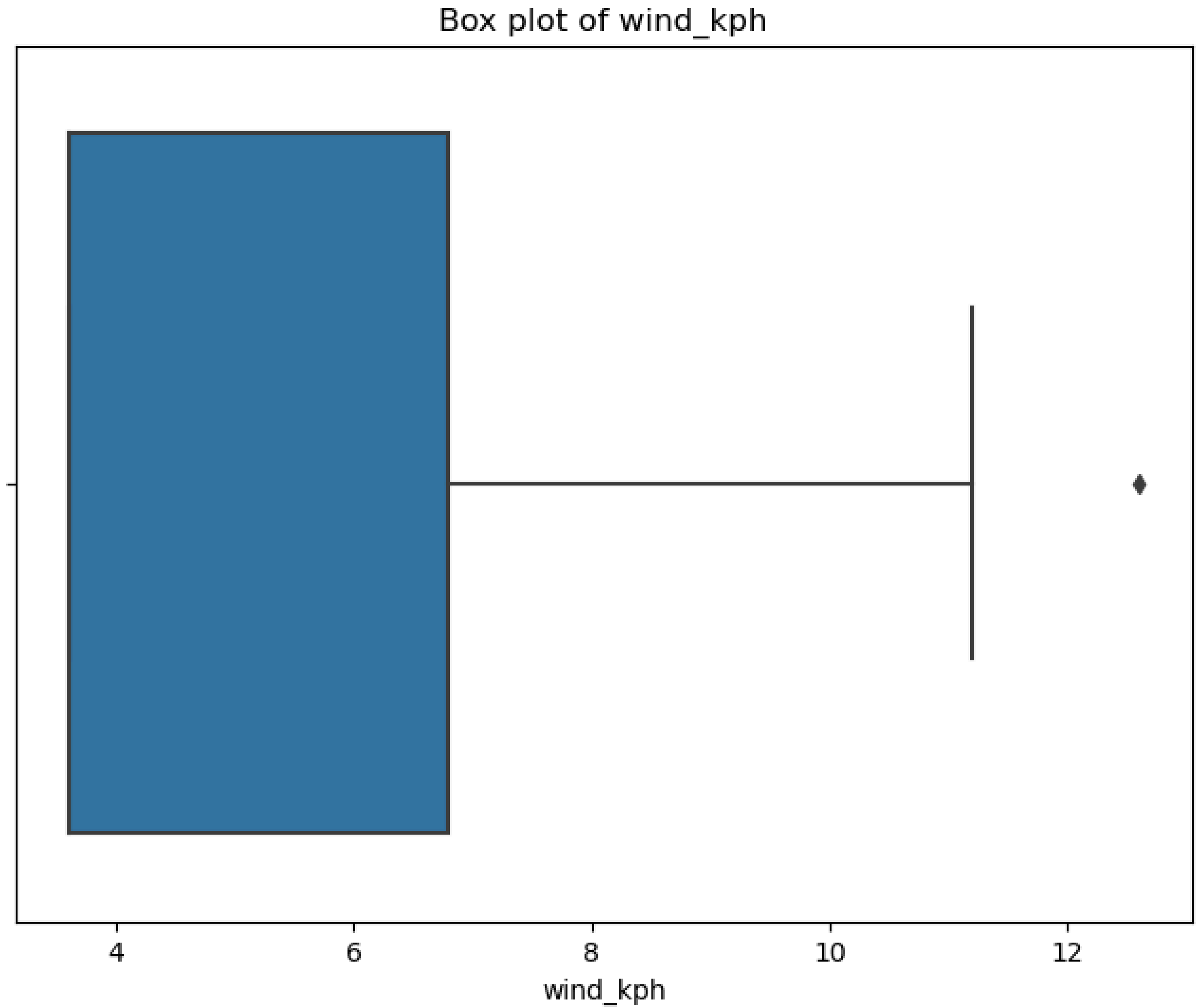
In [8]:
```
# Box plot : Detect the outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[numerical_columns])
plt.title("Box Plot")
plt.show()
```

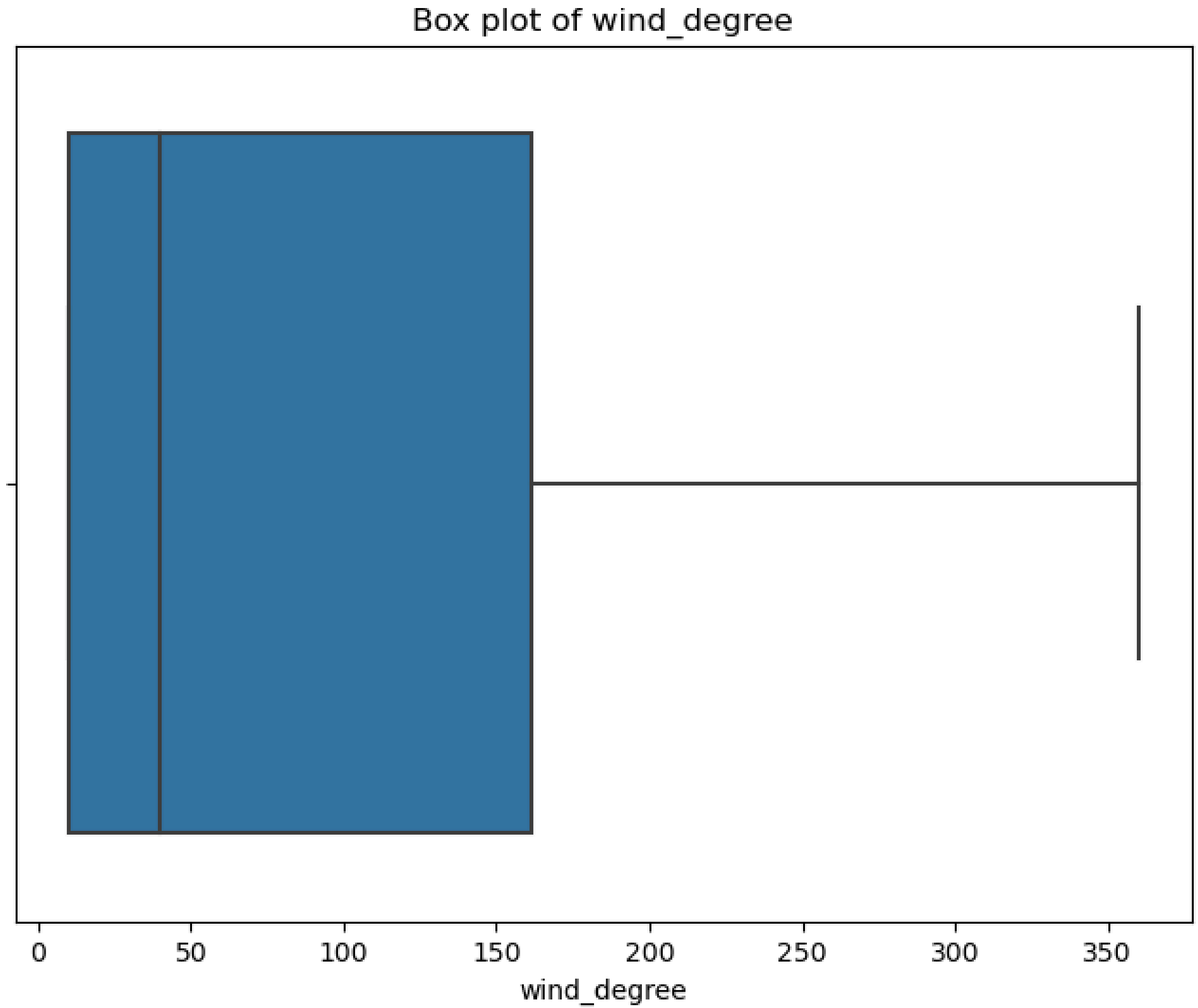


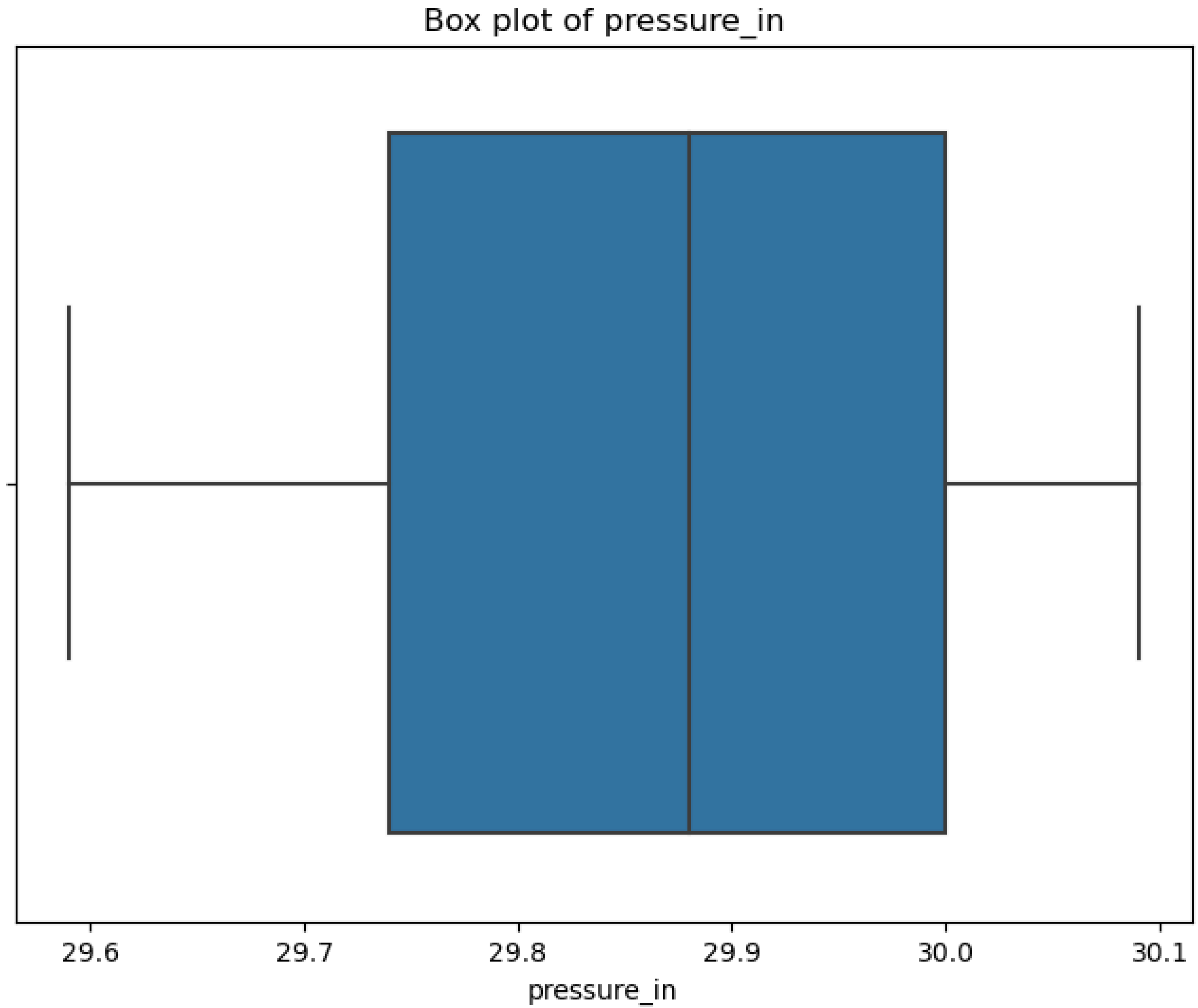
In [9]:

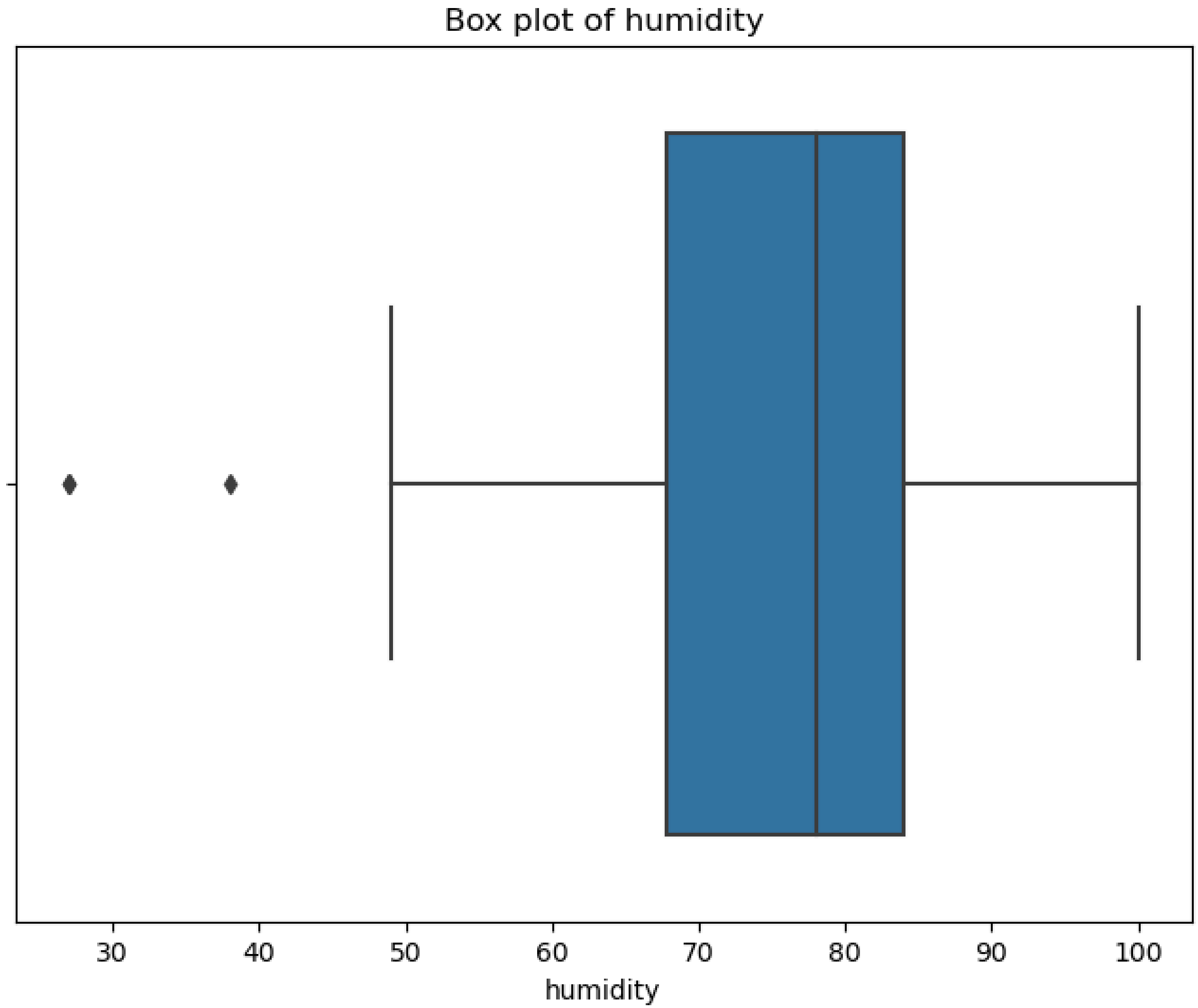
```
# Box plots
for column in numerical_columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=df[column])
    plt.title(f'Box plot of {column}')
    plt.xlabel(column)
    plt.show()
```

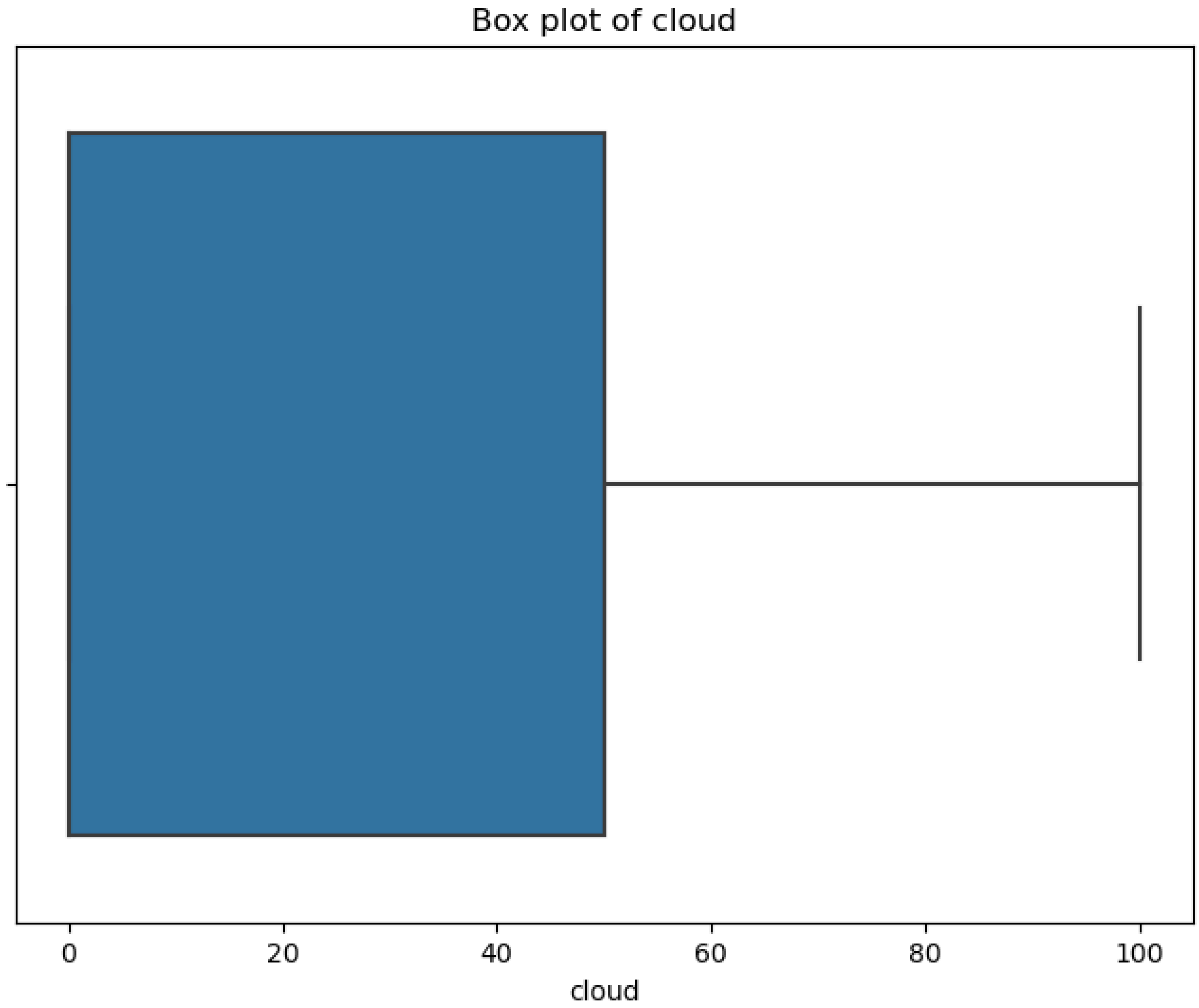

Box plot of temperature_celsius
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
temperature_celsius

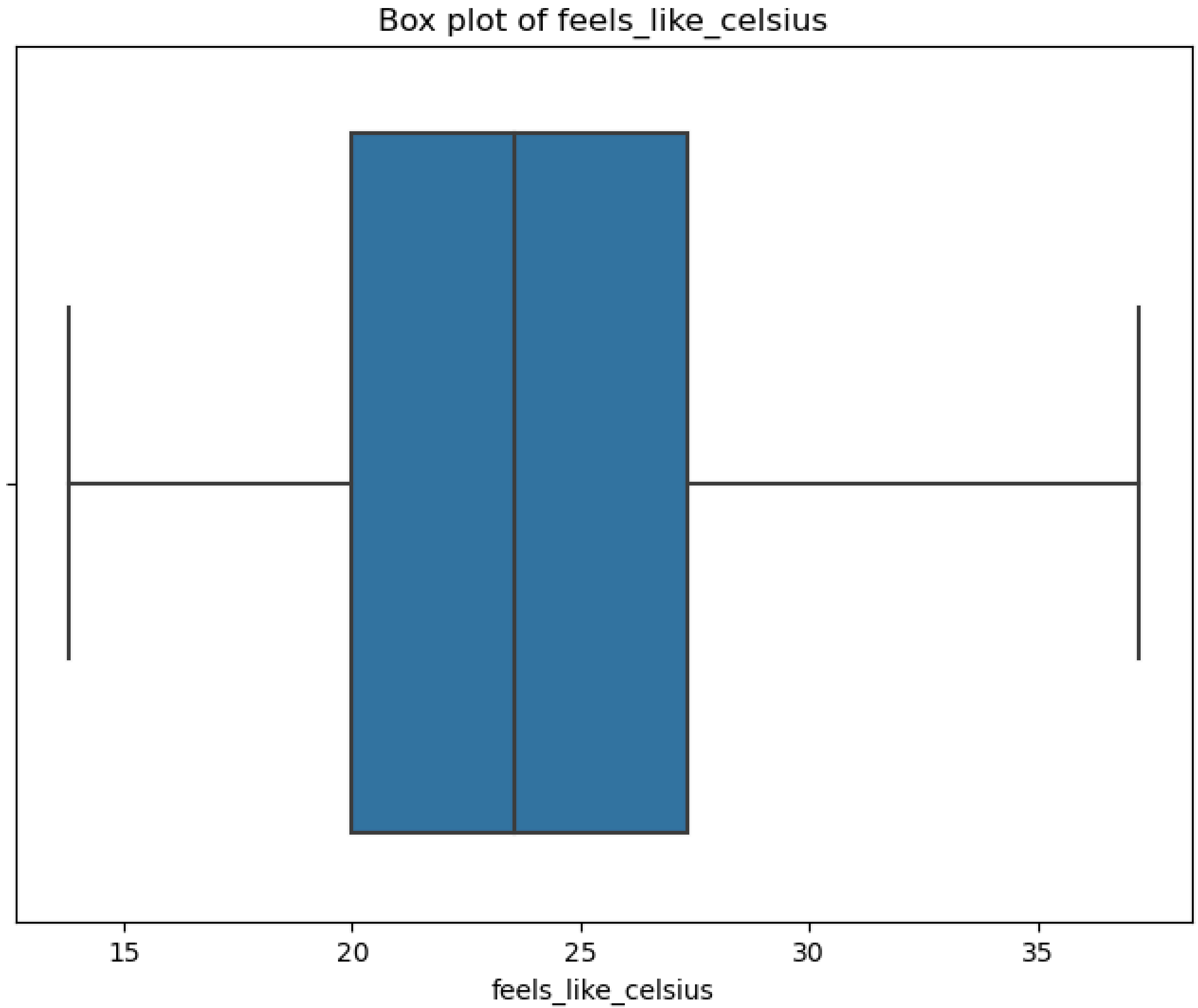

Box plot of wind_kph
4
6
8
10
12
wind_kph

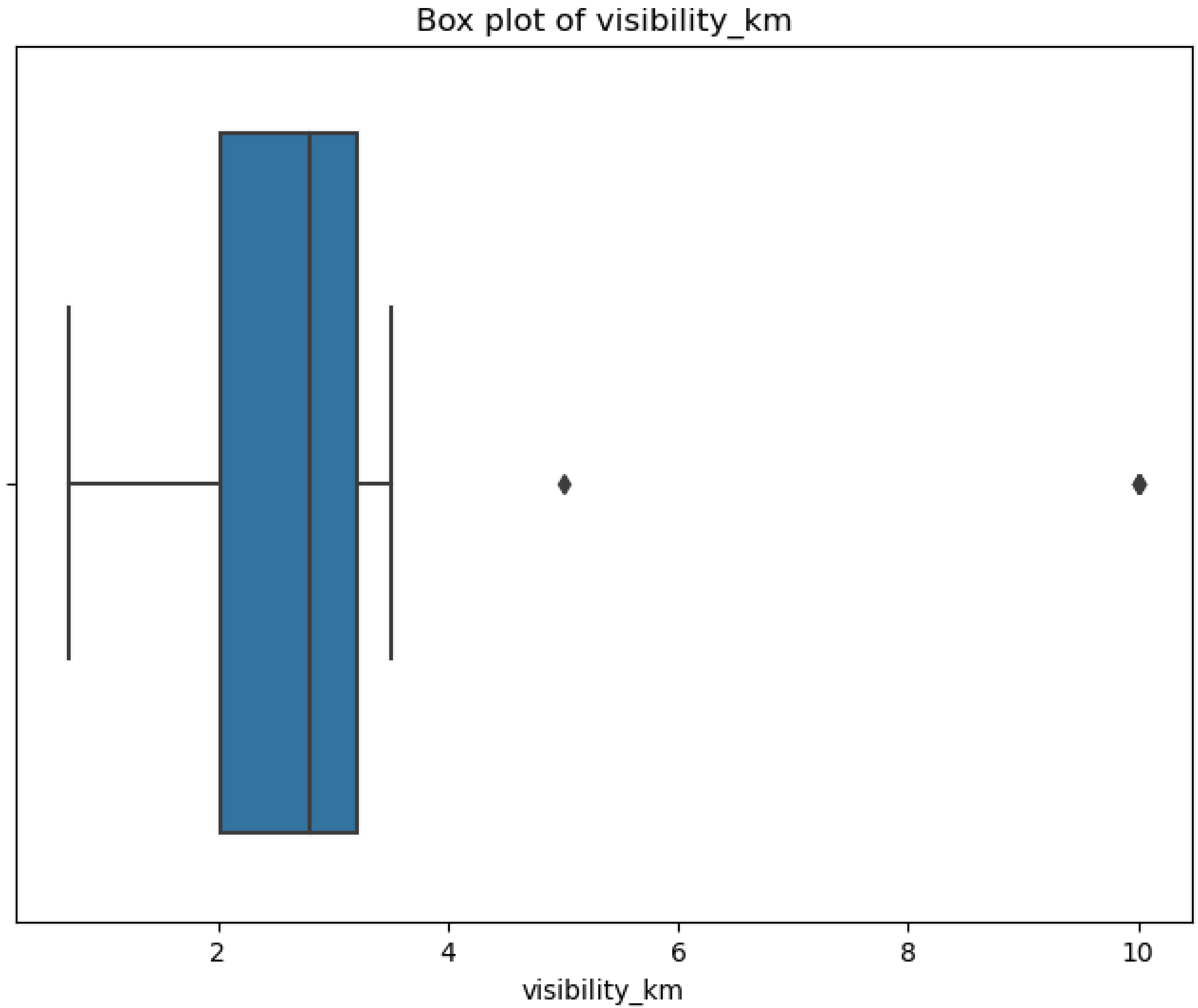

Box plot of wind_degree
0
50
100
150
200
250
300
350
wind_degree


Box plot of pressure_in
29.6
29.7
29.8
29.9
30.0
30.1
pressure_in


Box plot of humidity
30
40
50
60
70
80
90
100
humidity


Box plot of cloud
0
20
40
60
80
100
cloud


Box plot of feels_like_celsius
15
20
25
30
35
feels_like_celsius



Descriptive statistics

In [10]:
```
# Assuming your DataFrame is named 'df'
# You can use df.describe() for numerical columns and df[column].value_counts() for categorical columns
  # Summary statistics for numerical columns
df.describe()
```

Out[10]:

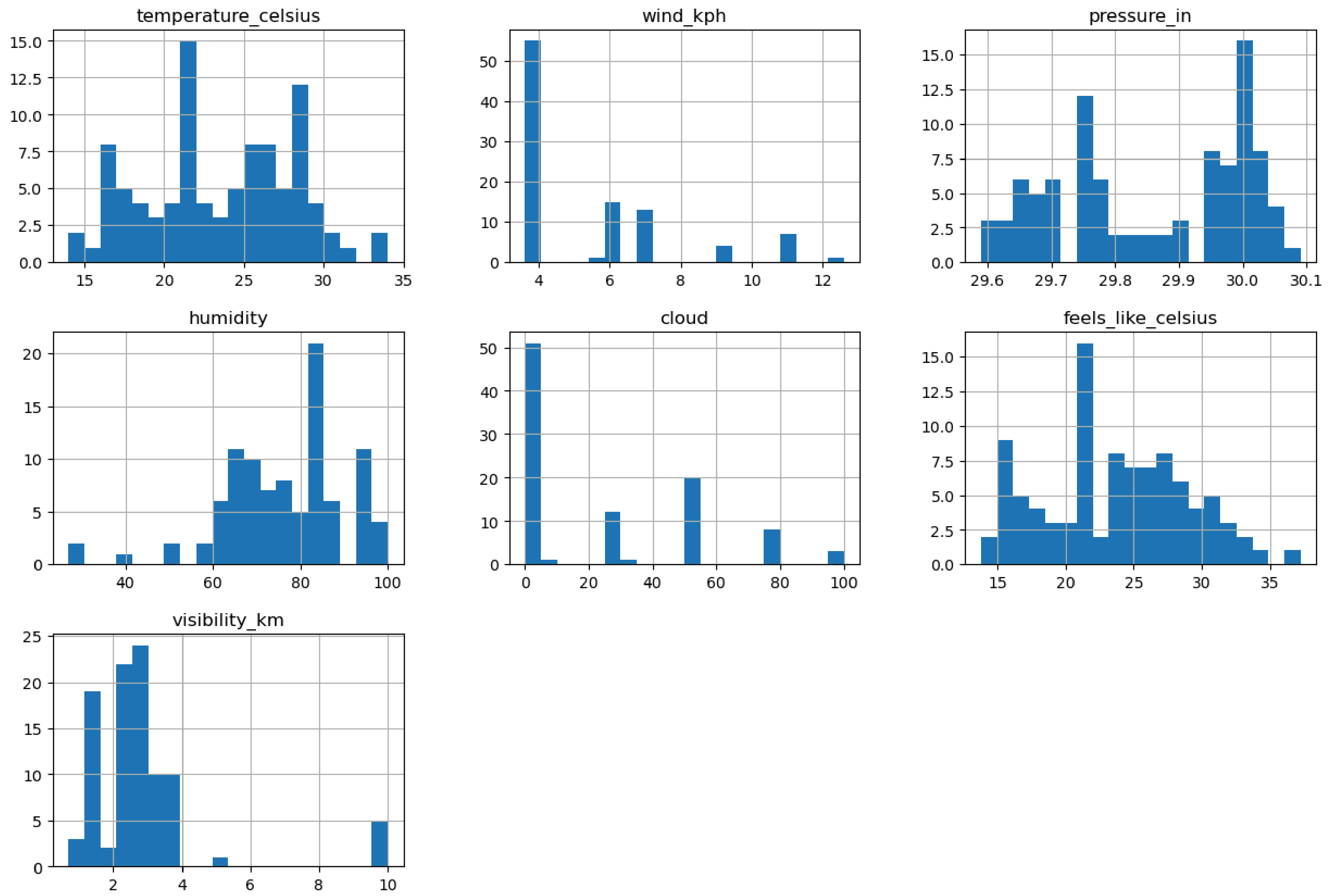
	temperature_celsius	wind_kph	wind_degree	pressure_in	humidity	cloud	feels_like_celsius	visibility_km
count	96.000000	96.000000	96.000000	96.000000	96.000000	96.000000	96.000000	96.000000
mean	23.044792	5.336458	97.239583	29.854167	76.218750	23.354167	23.604167	2.884375
std	4.666360	2.391657	109.834971	0.148109	14.327157	29.272577	5.283498	1.858031
min	14.000000	3.600000	10.000000	29.590000	27.000000	0.000000	13.800000	0.700000
25%	20.000000	3.600000	10.000000	29.740000	67.750000	0.000000	20.000000	2.025000
50%	23.000000	3.600000	40.000000	29.880000	78.000000	0.000000	23.550000	2.800000
75%	27.000000	6.800000	161.500000	30.000000	84.000000	50.000000	27.350000	3.200000
max	34.000000	12.600000	360.000000	30.090000	100.000000	100.000000	37.200000	10.000000

Check the distribution

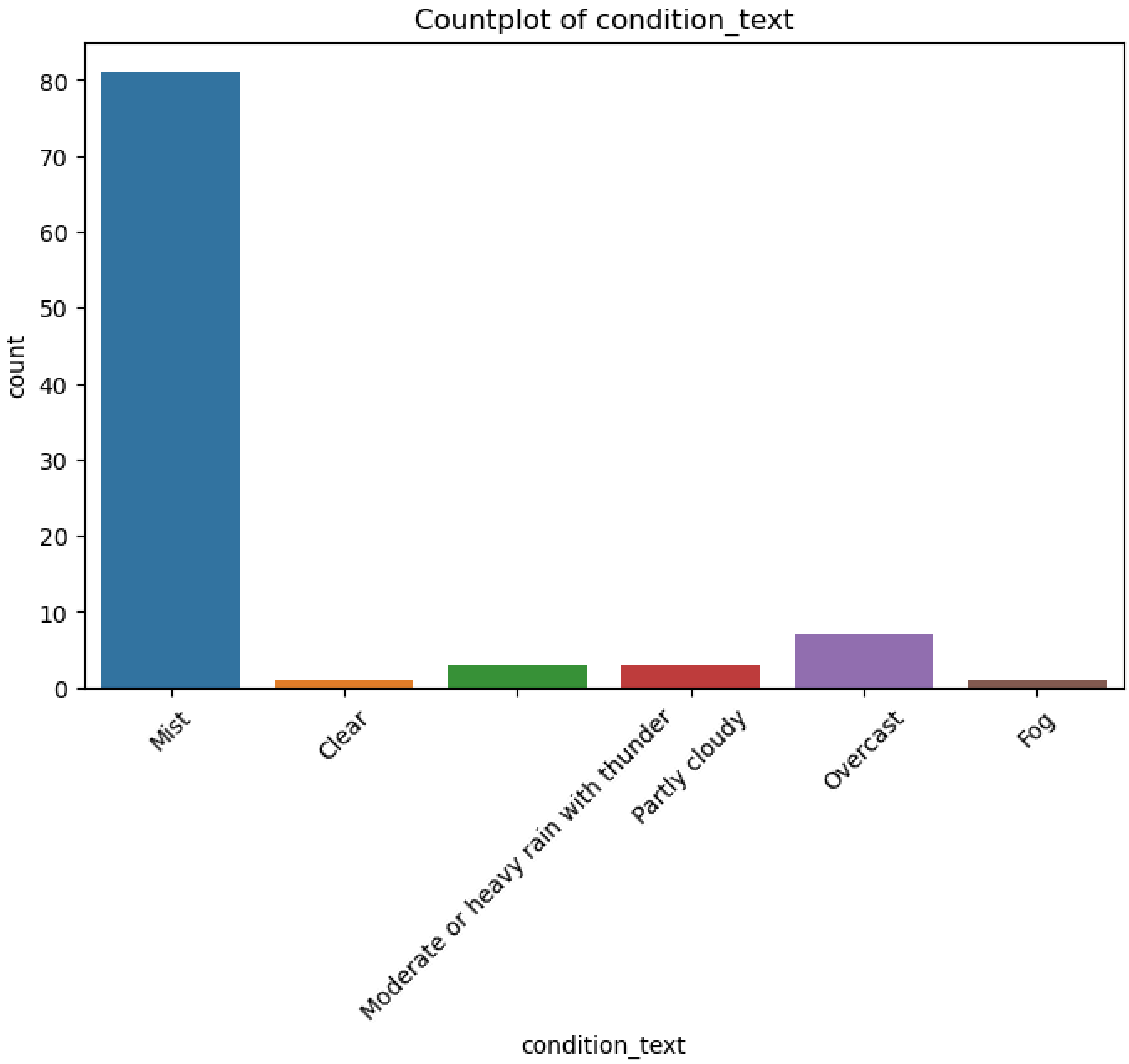
In [11]:

```
# Histograms for numerical columns
numerical_columns = ['temperature_celsius', 'wind_kph', 'pressure_in',
                     'humidity', 'cloud', 'feels_like_celsius', 'visibility_km']
df[numerical_columns].hist(bins=20, figsize=(15, 10))
plt.suptitle('Histograms of Numerical Columns')
plt.show()
```

Histograms of Numerical Columns



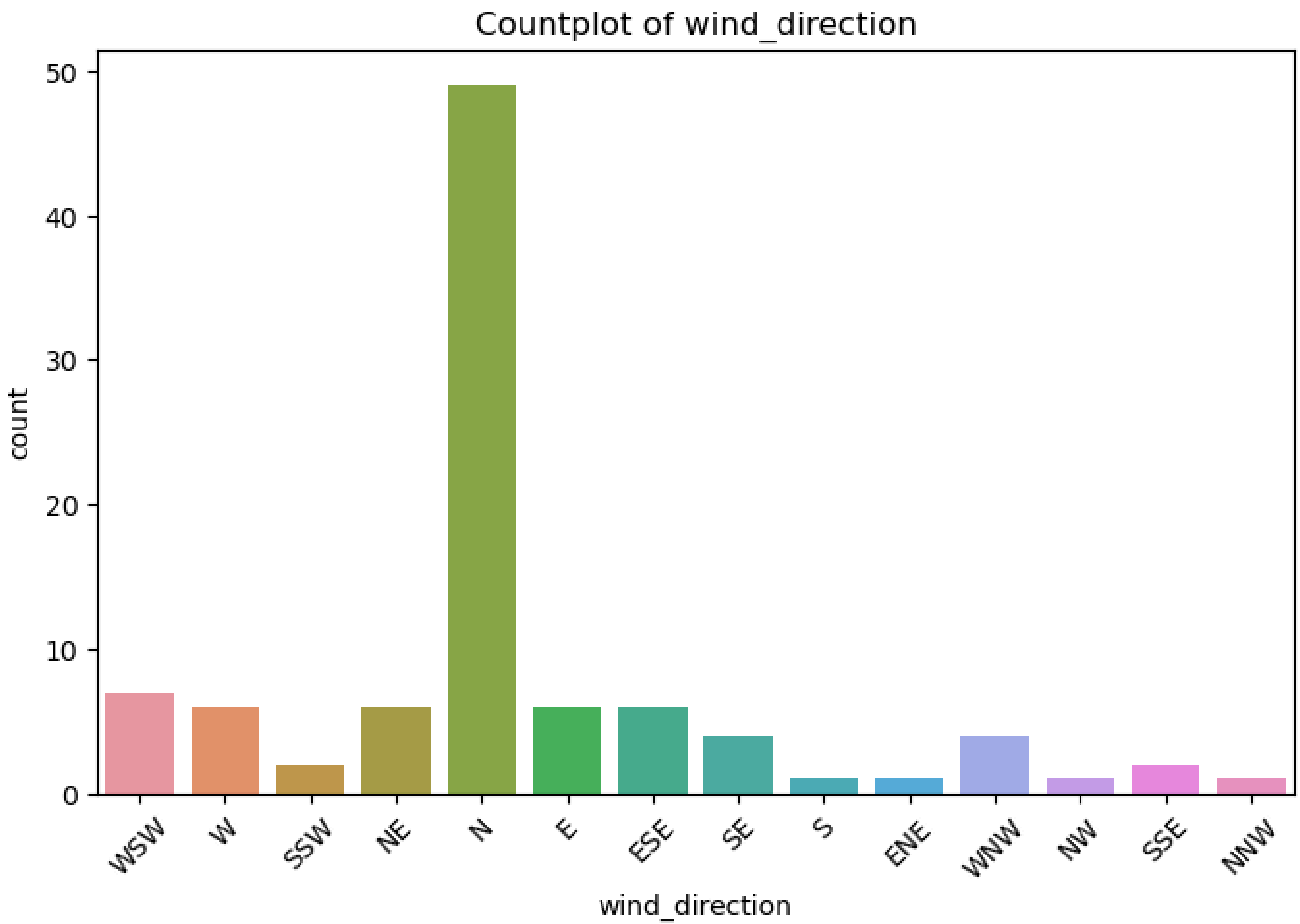
In [12]:
```
# Bar plots for categorical columns
categorical_columns = ['condition_text']
for column in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=df, x=column)
    plt.title(f'Countplot of {column}')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
    plt.show()
```



Interpretation :

From the above figure we can observe that the condition text is high in mist condition.

In [13]:
```
# Bar plots for categorical columns
categorical_columns = ['wind_direction']
for column in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=df, x=column)
    plt.title(f'Countplot of {column}')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
    plt.show()
```

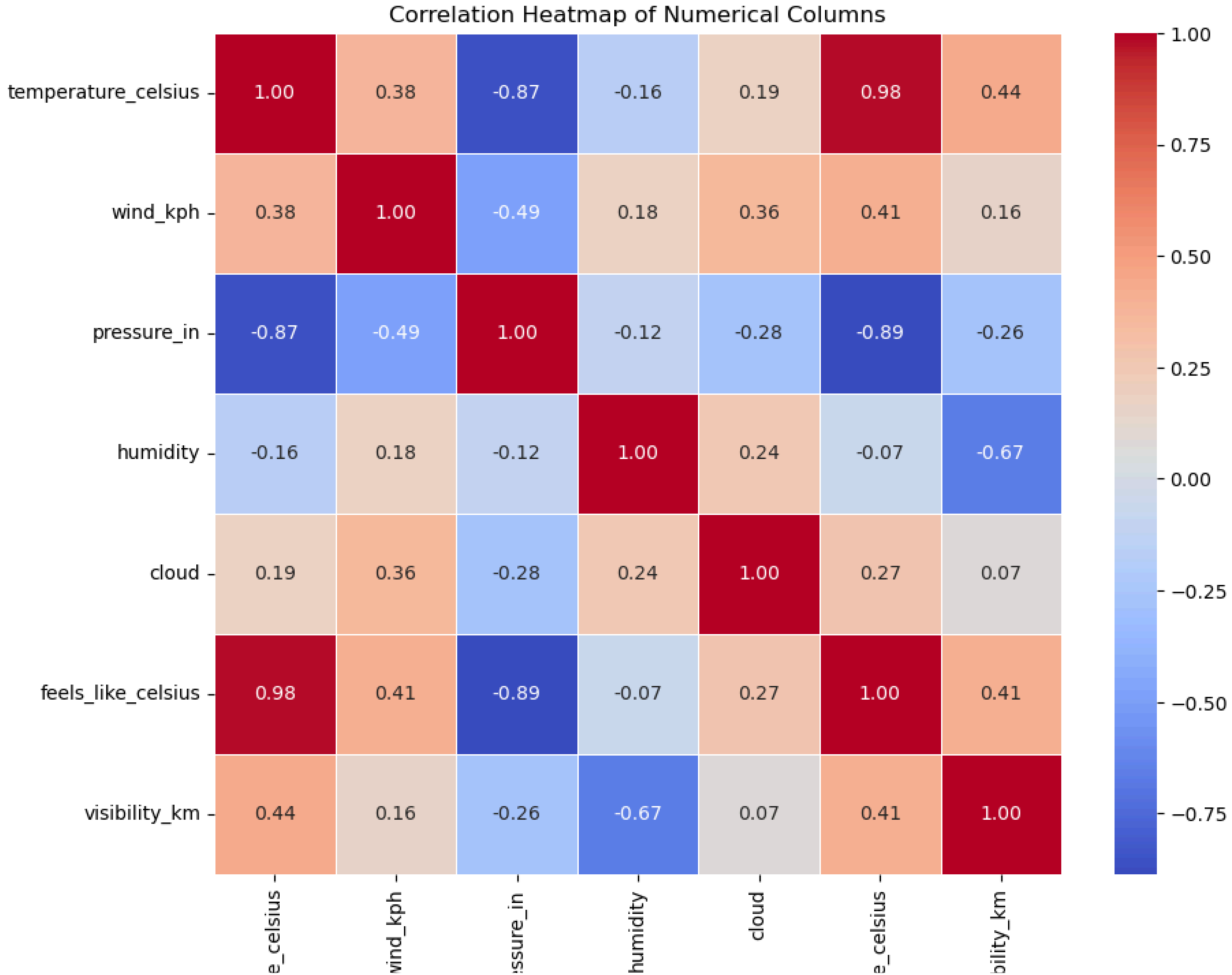


Interpretation :

From the above figure we can observe that the wind direction is high in North direction.

Correlation Heatmap

In [14]:
```
# Correlation matrix and heatmap for numerical columns
correlation_matrix = df[numerical_columns].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```

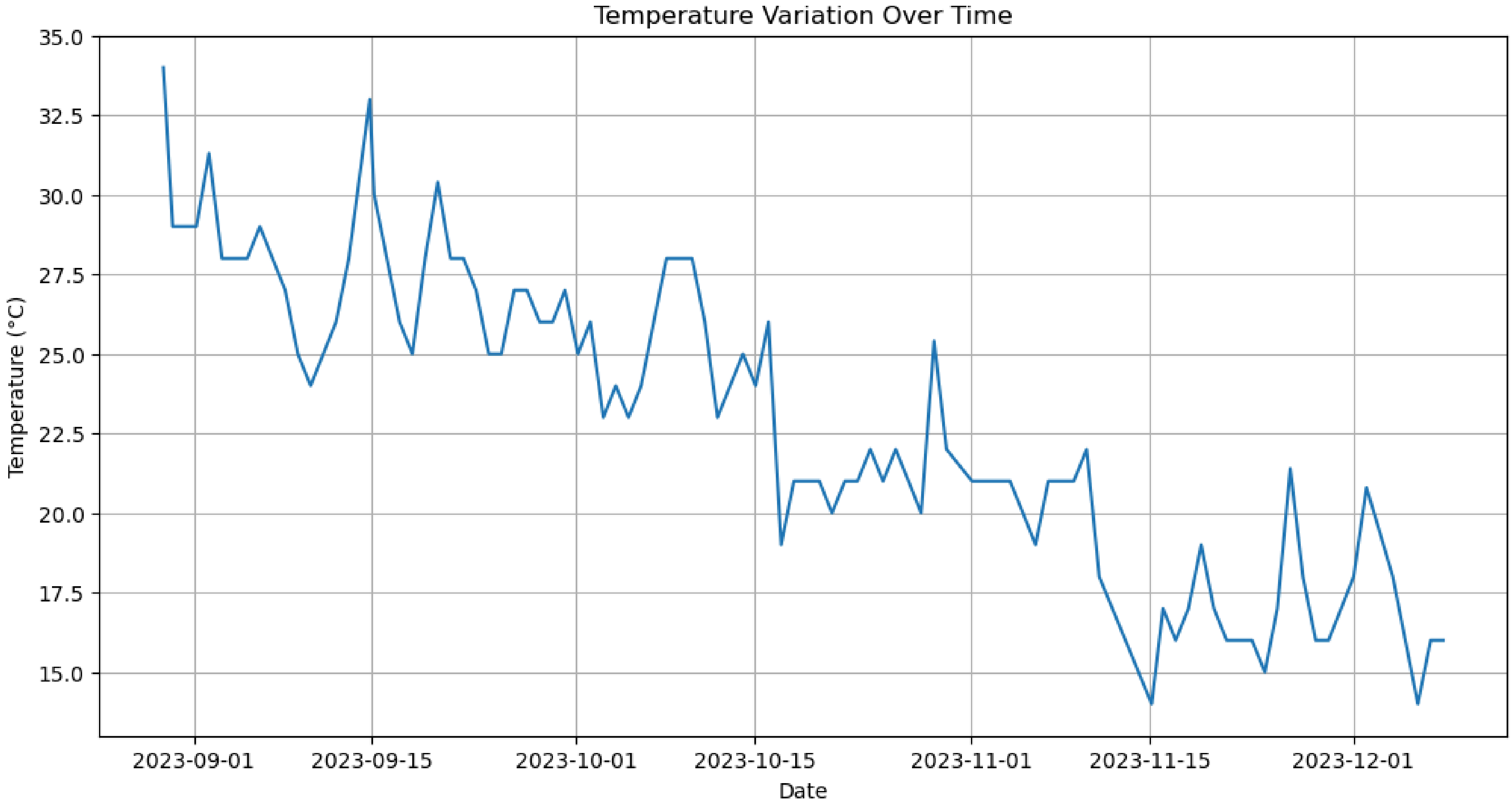

Correlation Heatmap of Numerical Columns
temperature_celsius
wind_kph
pressure_in
humidity
cloud
feels_like_celsius
visibility_km
1.00 0.38 -0.87 -0.16 0.19 0.98 0.44
0.38 1.00 -0.49 0.18 0.36 0.41 0.16
-0.87 -0.49 1.00 -0.12 -0.28 -0.89 -0.26
-0.16 0.18 -0.12 1.00 0.24 -0.07 -0.67
0.19 0.36 -0.28 0.24 1.00 0.27 0.07
0.98 0.41 -0.89 -0.07 0.27 1.00 0.41
0.44 0.16 -0.26 -0.67 0.07 0.41 1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Interpretation :

    From the above heatmap results we can see temperature_celsius is highly correlated with feels_like_celsius.

Line graph

In [15]:
```
# Assuming 'last_updated' is a timestamp column
df['last_updated'] = pd.to_datetime(df['last_updated'])
plt.figure(figsize=(12, 6))
plt.plot(df['last_updated'], df['temperature_celsius'], linestyle='-')
plt.title('Temperature Variation Over Time')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()
```



Interpretation :

```
From the above figure we can observe that the tempuratue is decrease.
```

Ordinal Logistic regression

```
Ordinal logistic regression is used to model the relationship between an ordered multilevel dependent
variable and independent variables. In the modeling, values of the dependent variable have a natural order or
```

ranking.

In [16]:
```
pip install mord
```

```
Requirement already satisfied: mord in c:\users\shweta\anaconda3\lib\site-packages (0.7)Note: you may need to restart the kernel to use updated packages.
```

Coding on condition text column

```
Clear = 0
Fog = 1
mist =2
Moderate or heavy rain with thunder = 3
overcast = 4
partly cloudy = 5
```

```
Here we use variable selection method for better result, we check which column condition text is related to this column by using R programming. Then we got 4 columns 'visibility_km', 'pressure_in', 'cloud', 'wind_kph'.
```

In [17]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from mord import LogisticAT
from sklearn.metrics import accuracy_score

# Load your dataset (assuming it's already loaded into weather_data DataFrame)
weather_data = pd.read_excel('coding(W data).xlsx')

# Define independent variables (X) and dependent variable (y)
X = weather_data[['visibility_km', 'pressure_in', 'cloud', 'wind_kph']]
y = weather_data['condition_text']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the ordinal logistic regression model
model = LogisticAT()

# Train the model
```

```
model.fit(X_train, y_train)

# Predict the target variable
y_pred = model.predict(X_test)

# Calculate accuracy
acc1 = accuracy_score(y_test, y_pred)
print("testing Accuracy:", acc1)
```

```
testing Accuracy: 0.8
```

Interpretation :

    Accuracy of ordinal logistic regression is 0.8%.

In [18]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Draw confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate evaluation measures
eval_report = classification_report(y_test, y_pred)
print("Evaluation Measures:")
print(eval_report)
```

```
Confusion Matrix:
[[ 0  0  0  1  0]
 [ 0 15  0  0  0]
 [ 0  1  0  0  0]
 [ 0  1  0  0  1]
 [ 0  0  0  0  1]]
Evaluation Measures:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           2       0.88      1.00      0.94        15
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         2
           5       0.50      1.00      0.67         1

    accuracy                           0.80        20
   macro avg       0.28      0.40      0.32        20
weighted avg       0.69      0.80      0.74        20
```

```
C:\Users\Shweta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shweta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shweta\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [19]:

```
import pandas as pd
from mord import LogisticAT

# Load your trained model (assuming it's already trained and saved)
# model = LogisticAT.load('your_model_path.pkl')

# Prepare input data for prediction
new_data = pd.DataFrame({
    'visibility_km': [10.5],  # Example visibility in kilometers
    'pressure_in': [29.5],      # Example pressure in inches
    'cloud': [3],               # Example cloud cover level
```

```
    'wind_kph': [15.0]          # Example wind speed in kilometers per hour
})

# Make predictions
predictions = model.predict(new_data)

# Print predictions
print("Predicted weather condition:", predictions)
```

```
Predicted weather condition: [3]
```

Interpretation :

```
From the above code, if we enter the values of visibility_km, pressure_in, cloud, wind_kph in our mind, then
this model give us the weather condition.
i.e.  Moderate or heavy rain with thunder = 3
```

ARIMA model

In [20]:
```
#Import required Libraries
import numpy as np, pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from numpy import log
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
from pandas import read_csv
import multiprocessing as mp

### Just to remove warnings to prettify the notebook.
import warnings
warnings.filterwarnings("ignore")
```

In [21]:
```
# Load the dataset
df = pd.read_excel('W Data.xlsx')
df.tail()
```

Out[21]:

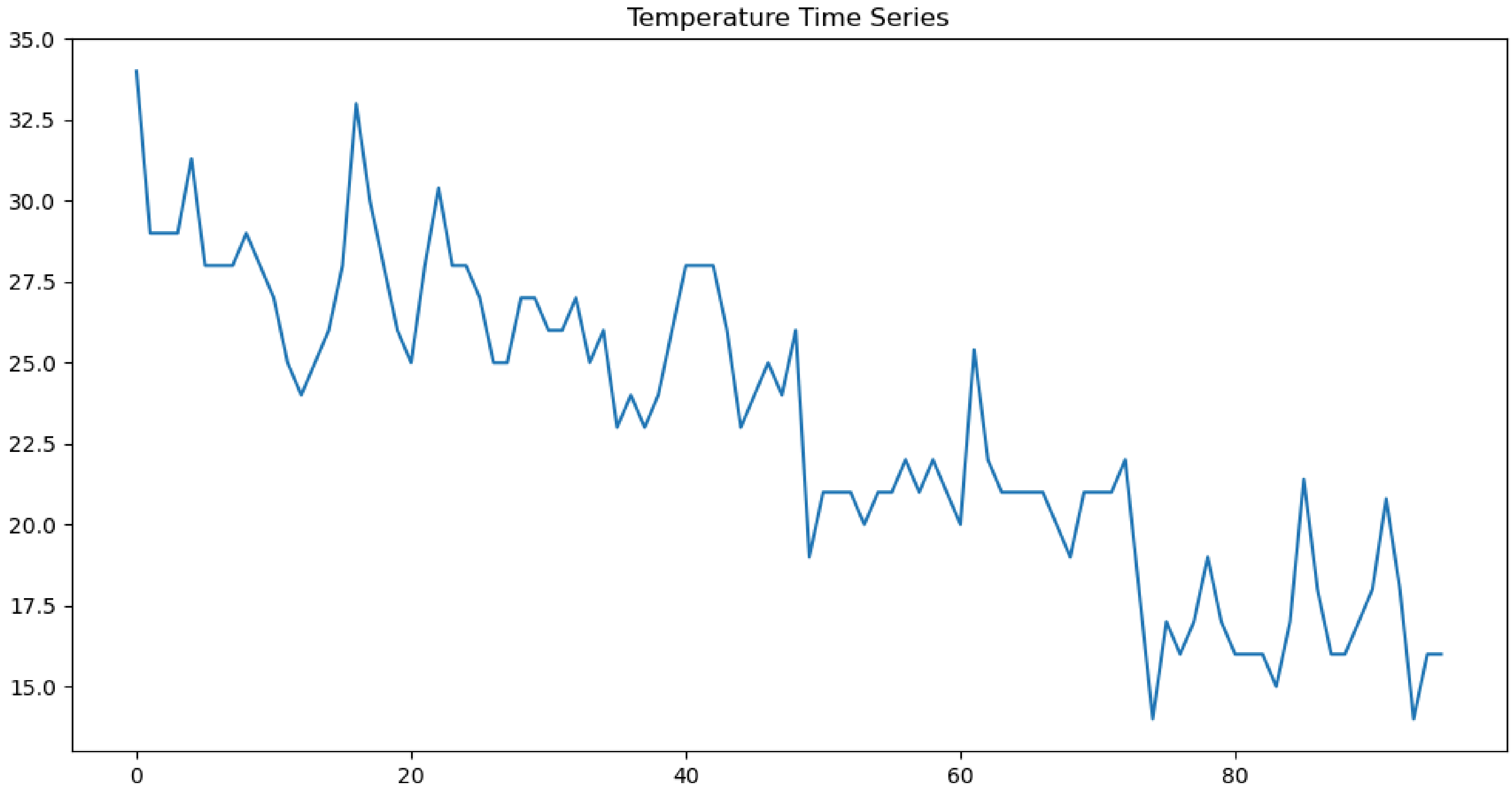
	country	location_name	timezone	last_updated	temperature_celsius	condition_text	wind_kph	wind_degree	wind_direction	pressure_in	humidit
91	India	New Delhi	Asia/Nicosia	2023-12-02 00:45:00	20.8	Partly cloudy	6.1	348	NNW	29.99	4
92	India	New Delhi	Asia/Nicosia	2023-12-04 02:00:00	18.0	Mist	3.6	10	N	30.03	8
93	India	New Delhi	Asia/Nicosia	2023-12-06 01:15:00	14.0	Mist	3.6	10	N	30.03	8
94	India	New Delhi	Asia/Nicosia	2023-12-07 01:15:00	16.0	Mist	3.6	10	N	30.03	7
95	India	New Delhi	Asia/Nicosia	2023-12-08 01:00:00	16.0	Mist	3.6	10	N	30.00	8

In [22]:

```
# Visualize the data
df['temperature_celsius'].plot(figsize=(12, 6), title='Temperature Time Series')
```

Out[22]:

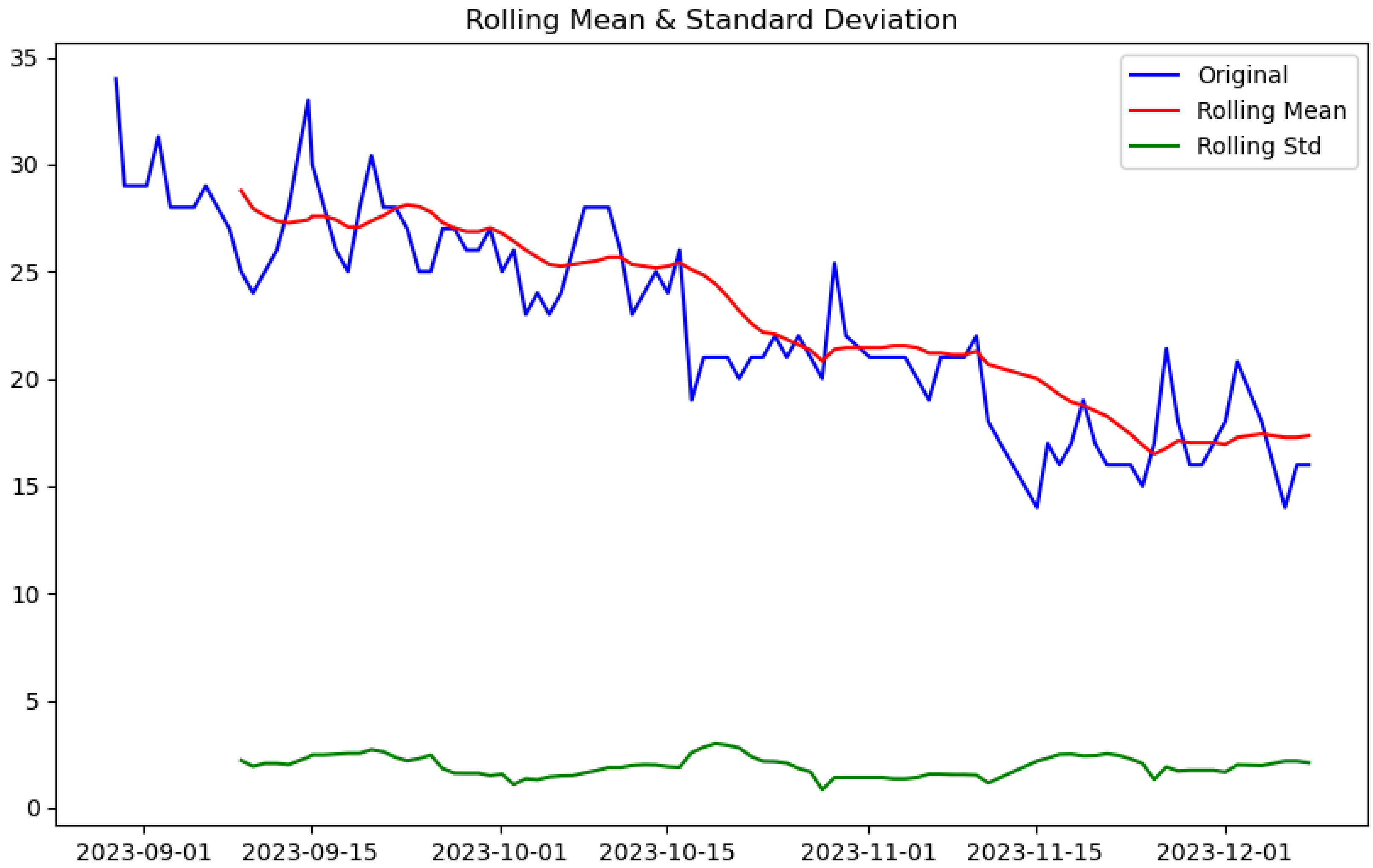
```
<AxesSubplot:title={'center':'Temperature Time Series'}>
```



In [23]:
```
# Set 'last_updated' as the index
df.set_index('last_updated', inplace=True)
```

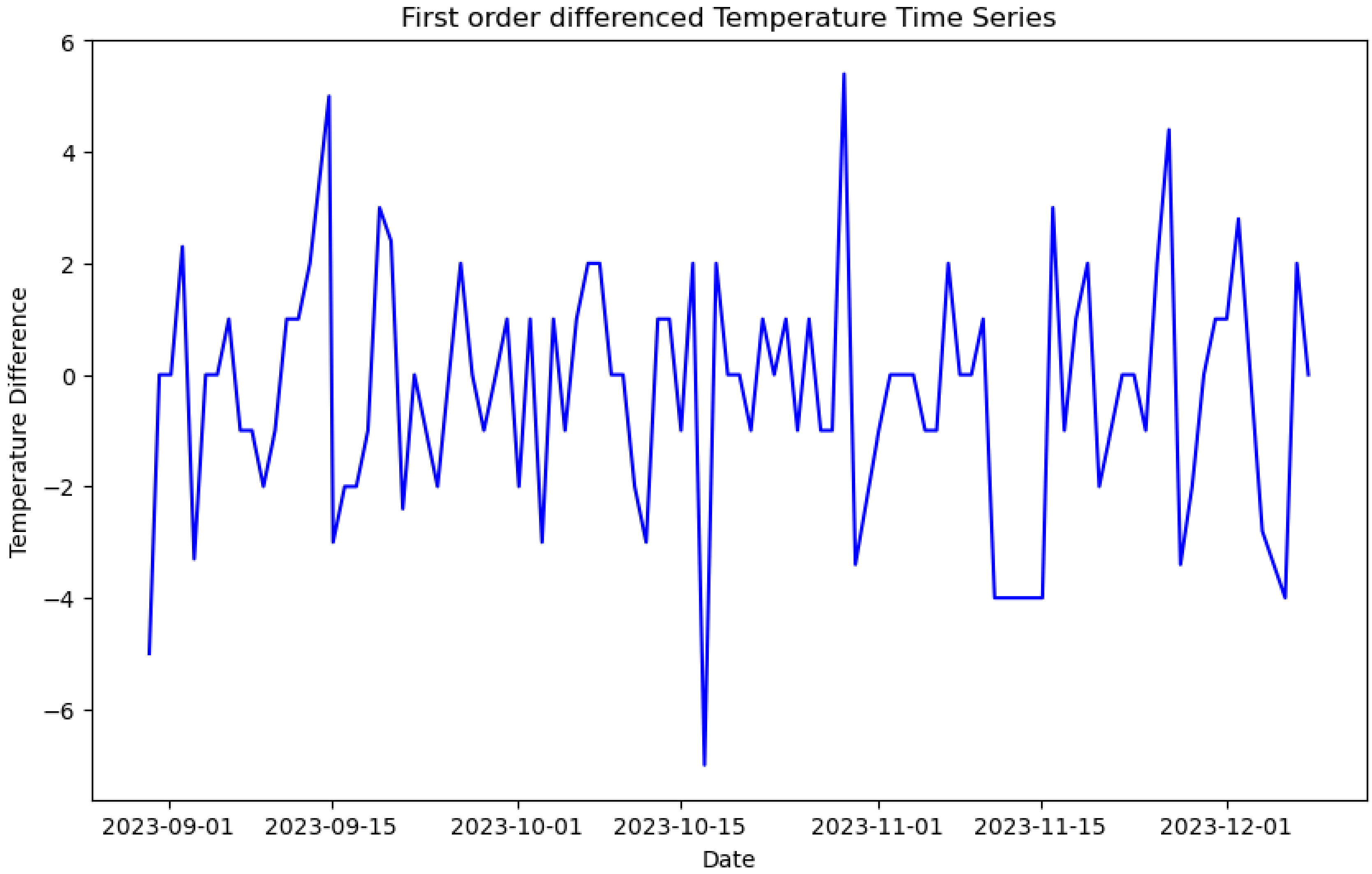
In [24]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Assuming 'df' is your DataFrame with time series data and 'last_updated' is already in datetime format

# Define a function for stationarity check
def check_stationarity(timeseries):
    # Calculate rolling statistics
```

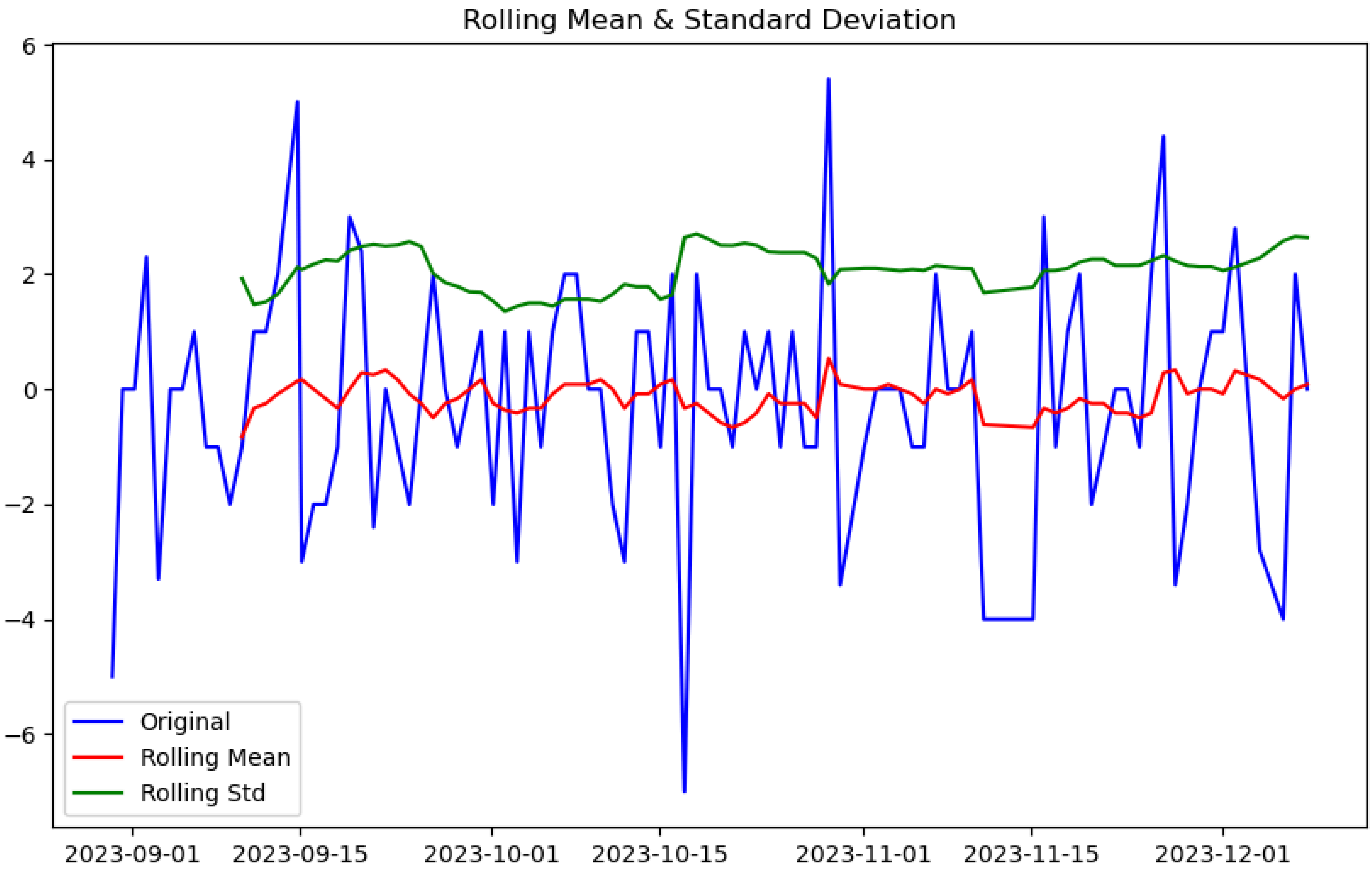
```
    rolling_mean = timeseries.rolling(window=12).mean()
    rolling_std = timeseries.rolling(window=12).std()

    # Plot rolling statistics
    plt.figure(figsize=(10, 6))
    plt.plot(timeseries, color='blue', label='Original')
    plt.plot(rolling_mean, color='red', label='Rolling Mean')
    plt.plot(rolling_std, color='green', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    # Perform Augmented Dickey-Fuller test
    adf_test = adfuller(timeseries, autolag='AIC')
    adf_results = pd.Series(adf_test[0:4], index=['Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used'])
    print('Augmented Dickey-Fuller Test:')
    print(adf_results)
    for key, value in adf_test[4].items():
        adf_results['Critical Value (%s)' %key] = value
    print(adf_results)

# Perform stationarity check on 'temperature_celsius' column
check_stationarity(df['temperature_celsius'])
```


Rolling Mean & Standard Deviation
Original
Rolling Mean
Rolling Std
35
30
25
20
15
10
5
0
2023-09-01
2023-09-15
2023-10-01
2023-10-15
2023-11-01
2023-11-15
2023-12-01

```
Augmented Dickey-Fuller Test:
Test Statistic                 -0.849706
p-value                         0.804084
#Lags Used                      5.000000
Number of Observations Used    90.000000
dtype: float64
Test Statistic                 -0.849706
p-value                         0.804084
#Lags Used                      5.000000
Number of Observations Used    90.000000
Critical Value (1%)            -3.505190
Critical Value (5%)            -2.894232
Critical Value (10%)           -2.584210
dtype: float64
```

In [25]:
```
df['temperature_diff'] = df['temperature_celsius'].diff()

df.dropna(inplace =True)

#plot differenced time series
plt.figure(figsize = (10,6))
plt.plot(df['temperature_diff'],color = 'blue')
plt.title('First order differenced Temperature Time Series')
plt.xlabel('Date')
plt.ylabel('Temperature Difference')
plt.show()
# perform stationarity check on differenced time series
check_stationarity(df['temperature_diff'])
```


First order differenced Temperature Time Series
Temperature Difference
Date
6
4
2
0
−2
−4
−6
2023-09-01
2023-09-15
2023-10-01
2023-10-15
2023-11-01
2023-11-15
2023-12-01


Rolling Mean & Standard Deviation
6
4
2
0
−2
−4
−6
Original
Rolling Mean
Rolling Std
2023-09-01
2023-09-15
2023-10-01
2023-10-15
2023-11-01
2023-11-15
2023-12-01

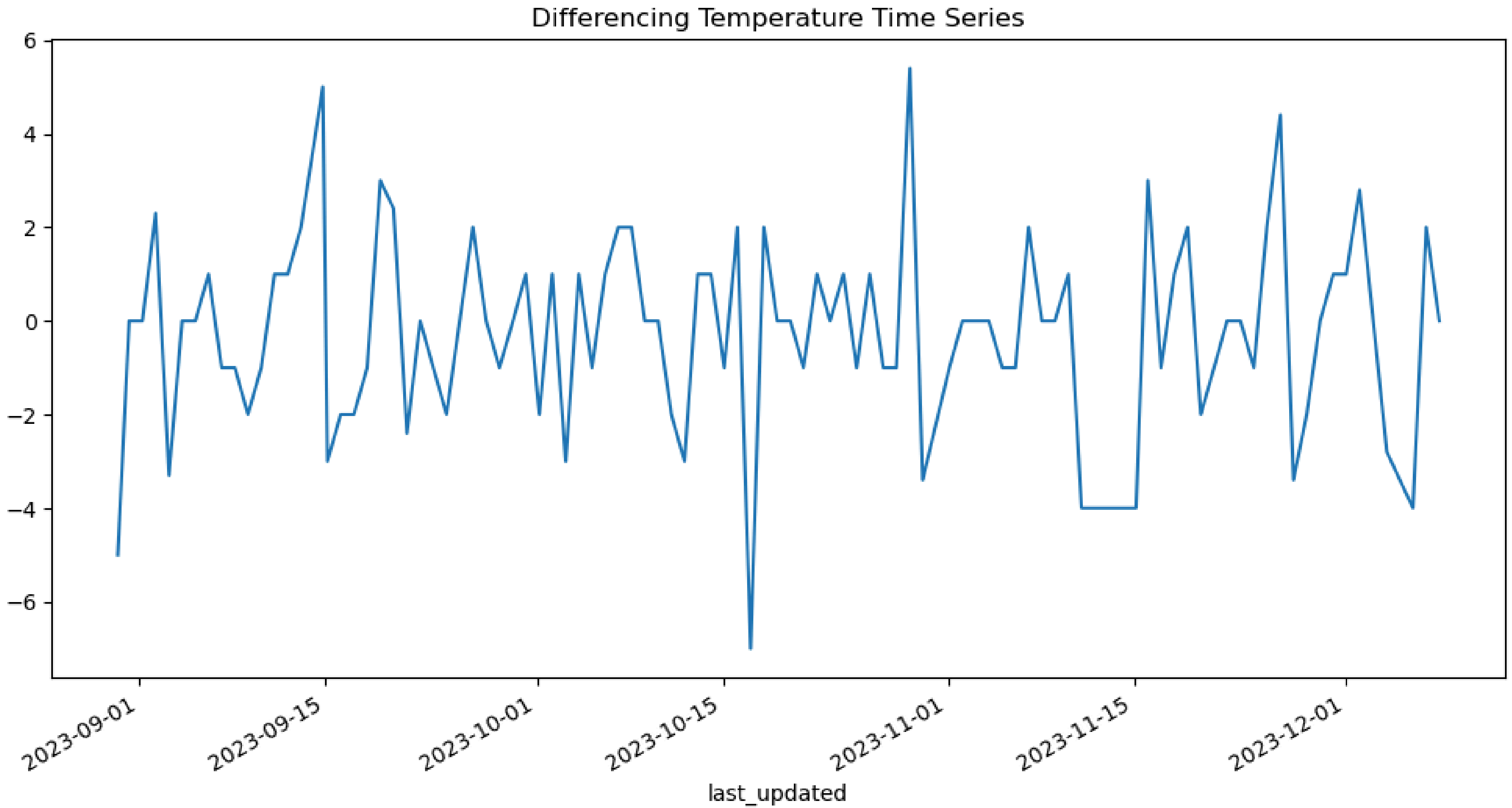
```
Augmented Dickey-Fuller Test:
Test Statistic                -8.075293e+00
p-value                        1.511215e-12
#Lags Used                     4.000000e+00
Number of Observations Used    9.000000e+01
dtype: float64
Test Statistic                -8.075293e+00
p-value                        1.511215e-12
#Lags Used                     4.000000e+00
Number of Observations Used    9.000000e+01
Critical Value (1%)           -3.505190e+00
Critical Value (5%)           -2.894232e+00
Critical Value (10%)          -2.584210e+00
dtype: float64
```

In [26]:

```
# Visualize the data
df['temperature_diff'].plot(figsize=(12, 6), title='Differencing Temperature Time Series')
```

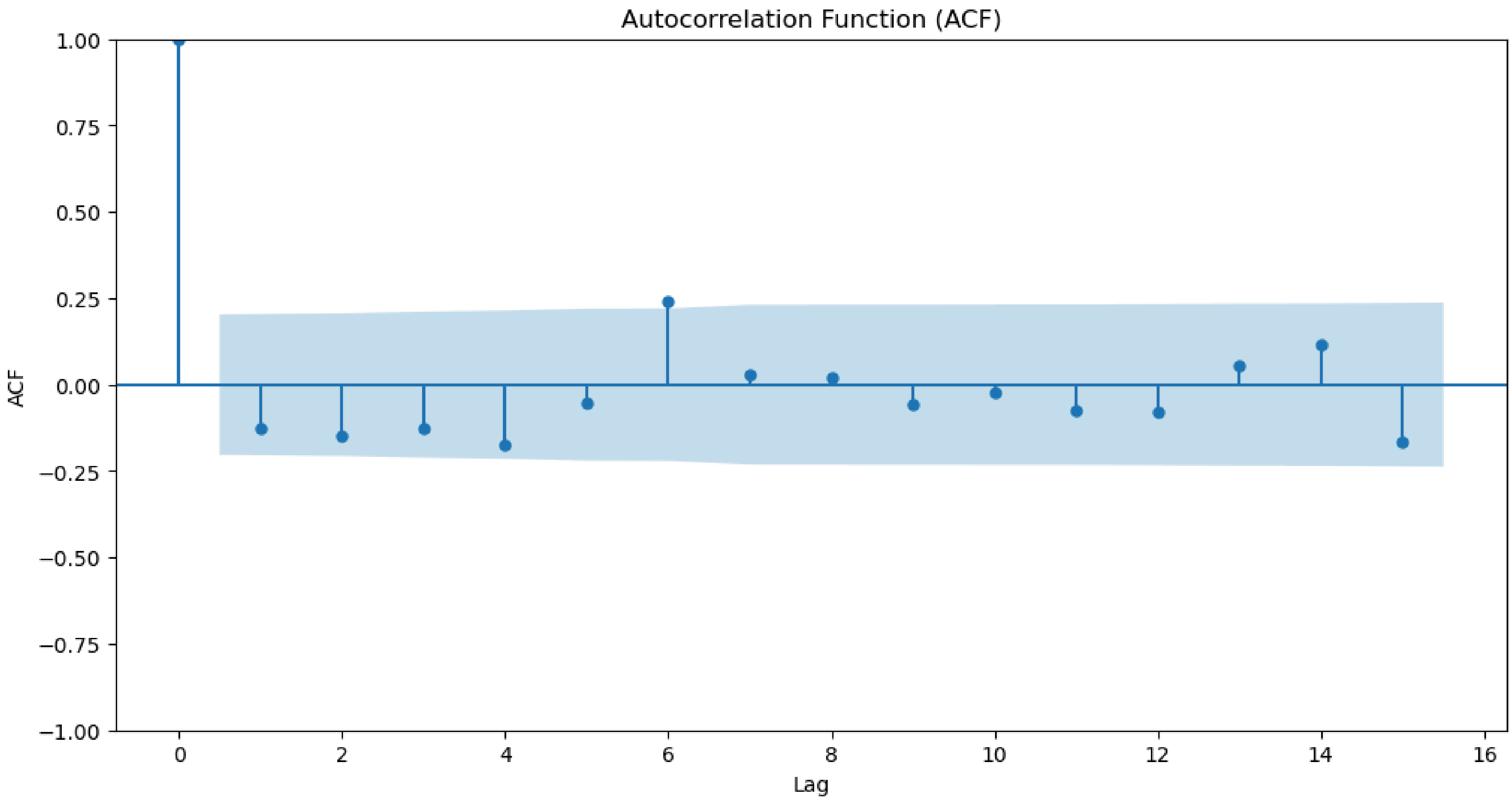
Out[26]:

```
<AxesSubplot:title={'center':'Differencing Temperature Time Series'}, xlabel='last_updated'>
```



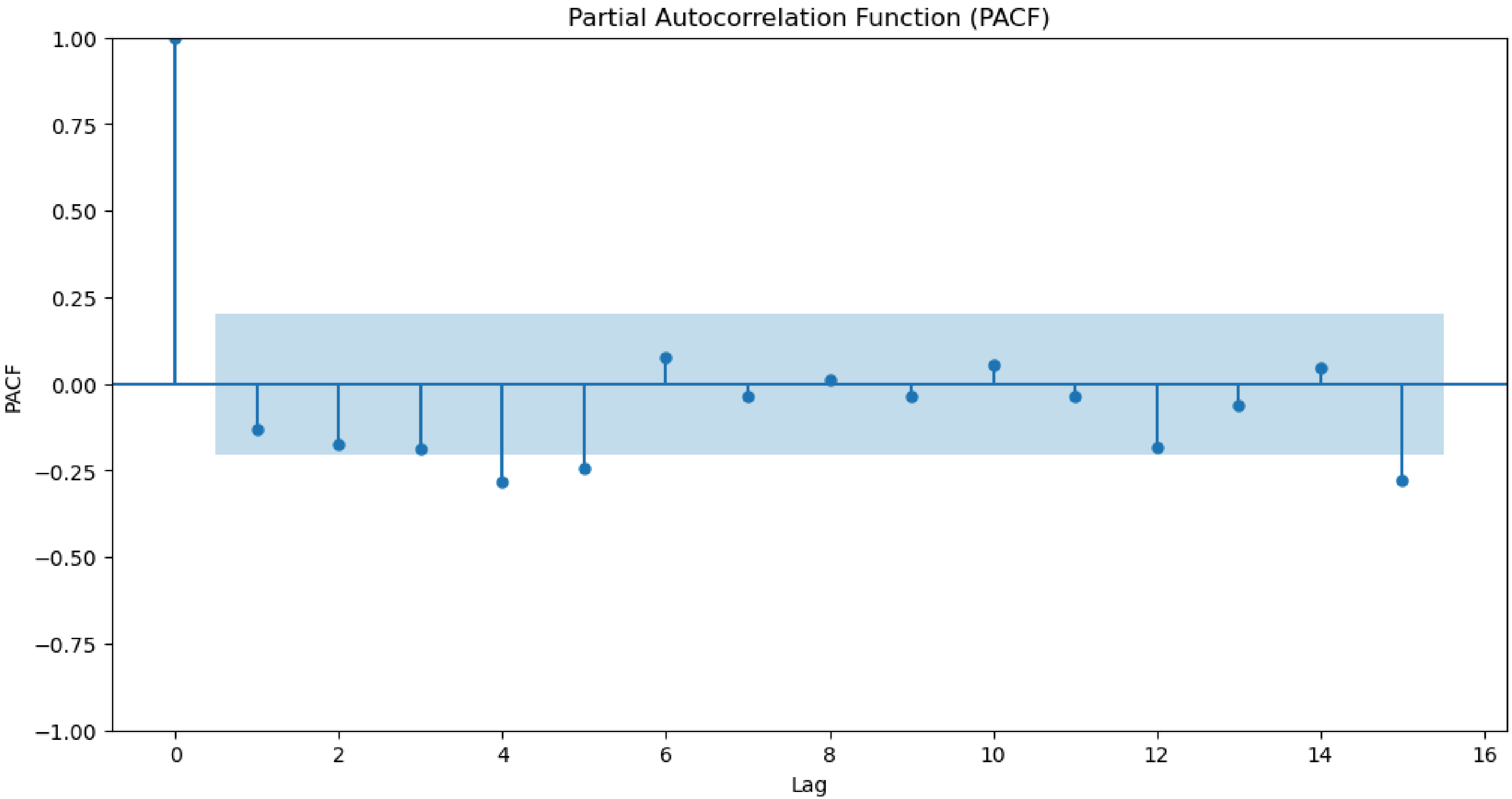
In [27]:
```
# Filter data within the specified date range
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

start_date = '2023-08-29'
end_date = '2023-12-07'
df_filtered = df[(df.index >= start_date) & (df.index <= end_date)]

# Plot ACF
plt.figure(figsize=(12, 6))
plot_acf(df_filtered['temperature_diff'],lags= 15 , ax=plt.gca())
plt.title('Autocorrelation Function (ACF)')
plt.xlabel('Lag')
```

```
plt.ylabel('ACF')
plt.show()
```



In [28]:
```
# Filter data within the specified date range
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
start_date = '2023-08-29'
end_date = '2023-12-07'
df_filtered = df[(df.index >= start_date) & (df.index <= end_date)]
# Plot ACF
plt.figure(figsize=(12, 6))
plot_pacf(df_filtered['temperature_diff'],lags= 15 , ax=plt.gca())
plt.title('Partial Autocorrelation Function (PACF)')
```

```
plt.xlabel('Lag')
plt.ylabel('PACF')
plt.show()
```



In [29]:
```
import statsmodels.api as sm
# Fit ARIMA model
arima_model = sm.tsa.ARIMA(df['temperature_diff'], order=(4,1,6))
arima_fit = arima_model.fit()
# Print summary of the fitted model
print(arima_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:       temperature_diff   No. Observations:                   95
Model:                 ARIMA(4, 1, 6)   Log Likelihood                -190.976
Date:                Sat, 13 Jul 2024   AIC                            403.951
Time:                        15:11:22   BIC                            431.927
Sample:                             0   HQIC                           415.251
                                 - 95
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.0968      0.157     -7.003      0.000      -1.404      -0.790
ar.L2         -1.2430      0.208     -5.962      0.000      -1.652      -0.834
ar.L3         -0.7391      0.194     -3.815      0.000      -1.119      -0.359
ar.L4         -0.5435      0.126     -4.323      0.000      -0.790      -0.297
ma.L1         -0.3655      2.760     -0.132      0.895      -5.775       5.044
ma.L2         -0.1255      3.497     -0.036      0.971      -6.980       6.729
ma.L3         -0.9264      2.330     -0.398      0.691      -5.493       3.640
ma.L4         -0.1555      3.825     -0.041      0.968      -7.652       7.341
ma.L5         -0.3957      1.975     -0.200      0.841      -4.266       3.475
ma.L6          0.9697      4.790      0.202      0.840      -8.419      10.358
sigma2         2.7226     13.413      0.203      0.839     -23.566      29.011
===================================================================================
Ljung-Box (L1) (Q):                   1.80   Jarque-Bera (JB):                 6.90
Prob(Q):                              0.18   Prob(JB):                         0.03
Heteroskedasticity (H):               0.85   Skew:                             0.02
Prob(H) (two-sided):                  0.66   Kurtosis:                         4.33
===================================================================================

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

In [30]:

```python
# Fit ARIMA model
arima_model = sm.tsa.ARIMA(df['temperature_diff'], order=(5,1,6))
arima_fit = arima_model.fit()

print(arima_fit.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:       temperature_diff   No. Observations:                   95
Model:                 ARIMA(5, 1, 6)   Log Likelihood                -192.434
Date:                Sat, 13 Jul 2024   AIC                            408.868
Time:                        15:11:22   BIC                            439.387
Sample:                             0   HQIC                           421.196
                                 - 95                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4410      0.348     -1.268      0.205      -1.123       0.241
ar.L2         -0.4823      0.354     -1.361      0.173      -1.177       0.212
ar.L3         -0.3439      0.296     -1.161      0.246      -0.924       0.237
ar.L4         -0.0271      0.290     -0.094      0.925      -0.595       0.541
ar.L5         -0.6714      0.200     -3.357      0.001      -1.063      -0.279
ma.L1         -0.8441      0.775     -1.089      0.276      -2.363       0.675
ma.L2          0.0102      0.324      0.032      0.975      -0.626       0.646
ma.L3         -0.2224      0.386     -0.576      0.565      -0.979       0.535
ma.L4         -0.2663      0.304     -0.875      0.381      -0.863       0.330
ma.L5          0.8935      0.476      1.876      0.061      -0.040       1.827
ma.L6         -0.5635      0.596     -0.946      0.344      -1.731       0.604
sigma2         3.2081      2.061      1.556      0.120      -0.832       7.248
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                14.49
Prob(Q):                              0.94   Prob(JB):                         0.00
Heteroskedasticity (H):               0.92   Skew:                            -0.01
Prob(H) (two-sided):                  0.81   Kurtosis:                         4.92
===================================================================================

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

Best Fit(4,1,0)

In [31]:

```
import statsmodels.api as sm
# Fit ARIMA model
arima_model = sm.tsa.ARIMA(df['temperature_diff'], order=(4,1,0))
arima_fit = arima_model.fit()
print(arima_fit.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:       temperature_diff   No. Observations:                   95
Model:                 ARIMA(4, 1, 0)   Log Likelihood                -215.781
Date:                Sat, 13 Jul 2024   AIC                            441.561
Time:                        15:11:22   BIC                            454.278
Sample:                             0   HQIC                           446.698
                                 - 95                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8039      0.089     -9.056      0.000      -0.978      -0.630
ar.L2         -0.6513      0.129     -5.030      0.000      -0.905      -0.397
ar.L3         -0.4366      0.146     -2.984      0.003      -0.723      -0.150
ar.L4         -0.2837      0.111     -2.559      0.010      -0.501      -0.066
sigma2         5.7097      0.967      5.908      0.000       3.815       7.604
===================================================================================
Ljung-Box (L1) (Q):                   1.06   Jarque-Bera (JB):                 0.68
Prob(Q):                              0.30   Prob(JB):                         0.71
Heteroskedasticity (H):               1.05   Skew:                            -0.18
Prob(H) (two-sided):                  0.90   Kurtosis:                         2.78
===================================================================================

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

In [32]:

```python
# Fit ARIMA model
arima_model = sm.tsa.ARIMA(df['temperature_diff'], order=(5,1,0))
arima_fit = arima_model.fit()
print(arima_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:       temperature_diff   No. Observations:                   95
Model:                 ARIMA(5, 1, 0)   Log Likelihood                -205.450
Date:                Sat, 13 Jul 2024   AIC                            422.899
Time:                        15:11:22   BIC                            438.159
Sample:                             0   HQIC                           429.063
                                 - 95
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.9333      0.078    -11.984      0.000      -1.086      -0.781
ar.L2         -0.8280      0.130     -6.362      0.000      -1.083      -0.573
ar.L3         -0.7222      0.143     -5.054      0.000      -1.002      -0.442
ar.L4         -0.6566      0.129     -5.106      0.000      -0.909      -0.405
ar.L5         -0.4687      0.100     -4.699      0.000      -0.664      -0.273
sigma2         4.5243      0.680      6.653      0.000       3.191       5.857
===================================================================================
Ljung-Box (L1) (Q):                   0.64   Jarque-Bera (JB):                 0.16
Prob(Q):                              0.43   Prob(JB):                         0.92
Heteroskedasticity (H):               1.02   Skew:                             0.05
Prob(H) (two-sided):                  0.97   Kurtosis:                         3.18
===================================================================================

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

In [33]:

```
best_params = (4,1,0)
```

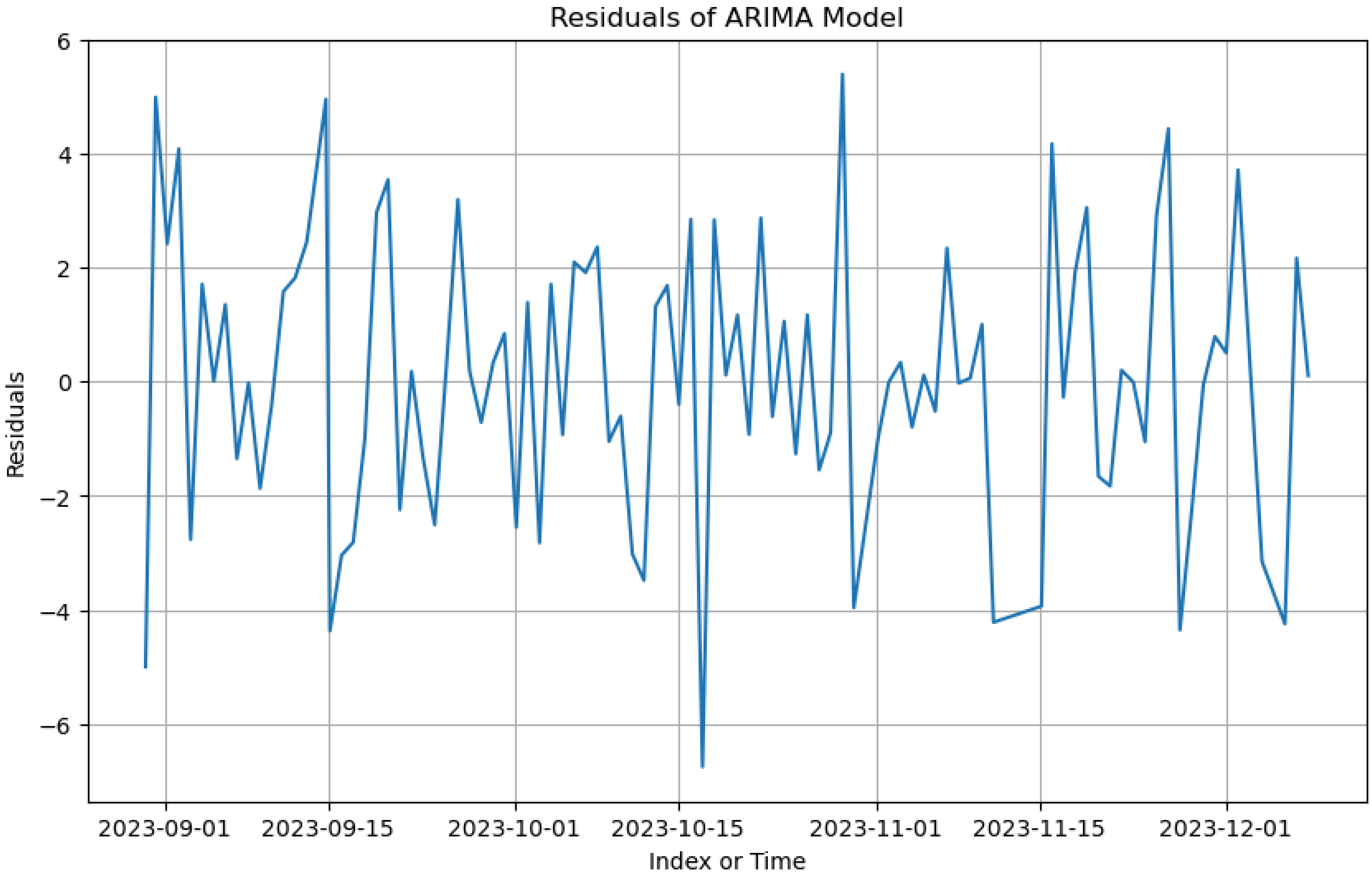
In [34]:

```
# Fit ARIMA model
arima_model = sm.tsa.ARIMA(df['temperature_diff'], order=best_params)
arima_fit = arima_model.fit()
print(arima_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:       temperature_diff   No. Observations:                   95
Model:                 ARIMA(4, 1, 0)   Log Likelihood                -215.781
Date:                Sat, 13 Jul 2024   AIC                            441.561
Time:                        15:11:22   BIC                            454.278
Sample:                             0   HQIC                           446.698
                                 - 95
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8039      0.089     -9.056      0.000      -0.978      -0.630
ar.L2         -0.6513      0.129     -5.030      0.000      -0.905      -0.397
ar.L3         -0.4366      0.146     -2.984      0.003      -0.723      -0.150
ar.L4         -0.2837      0.111     -2.559      0.010      -0.501      -0.066
sigma2         5.7097      0.967      5.908      0.000       3.815       7.604
===================================================================================
Ljung-Box (L1) (Q):                   1.06   Jarque-Bera (JB):                 0.68
Prob(Q):                              0.30   Prob(JB):                         0.71
Heteroskedasticity (H):               1.05   Skew:                            -0.18
Prob(H) (two-sided):                  0.90   Kurtosis:                         2.78
===================================================================================

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

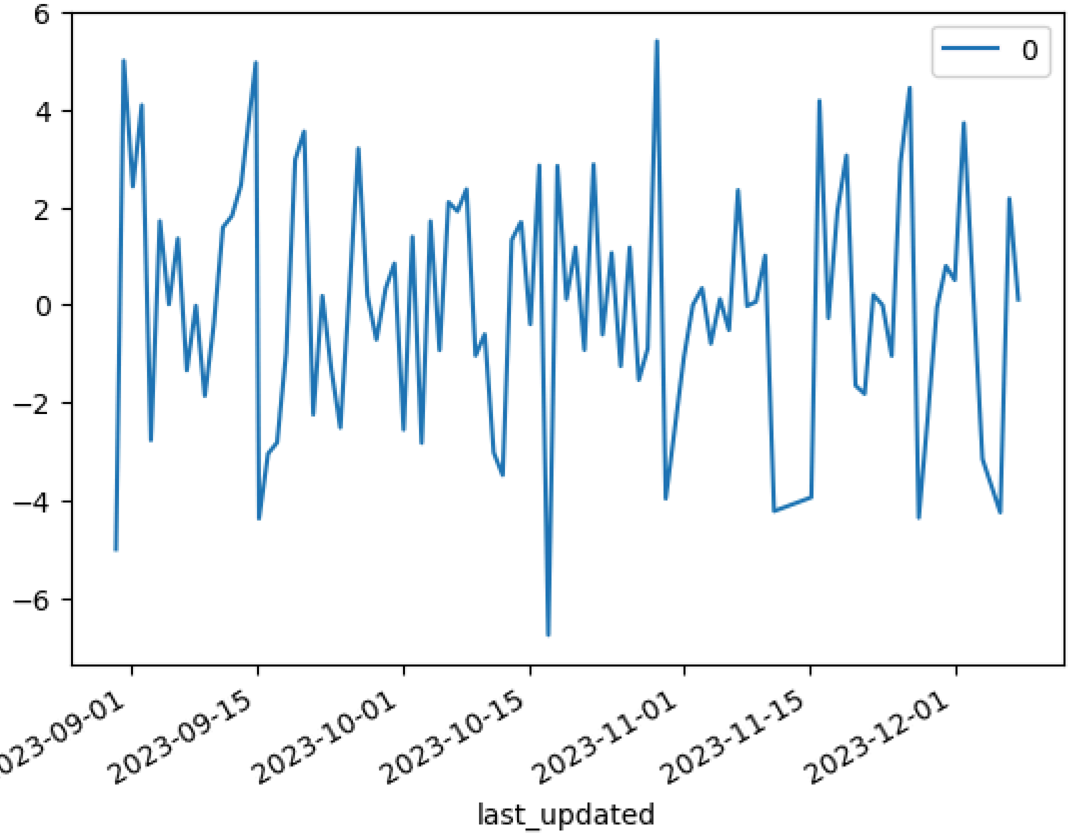
In [35]:

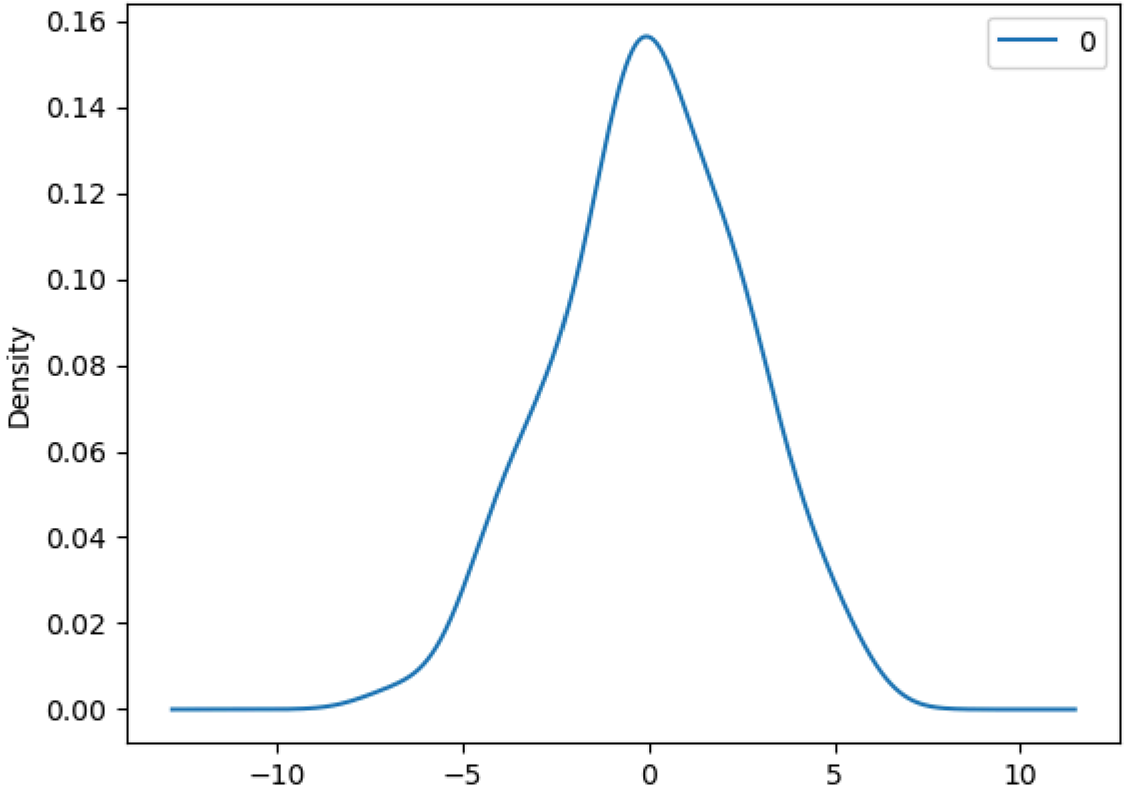
```python
import matplotlib.pyplot as plt
# Get the residuals from the ARIMA model
residuals = arima_fit.resid
# Plot the residuals
plt.figure(figsize=(10, 6))
plt.plot(residuals)
plt.title('Residuals of ARIMA Model')
plt.xlabel('Index or Time')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()
```



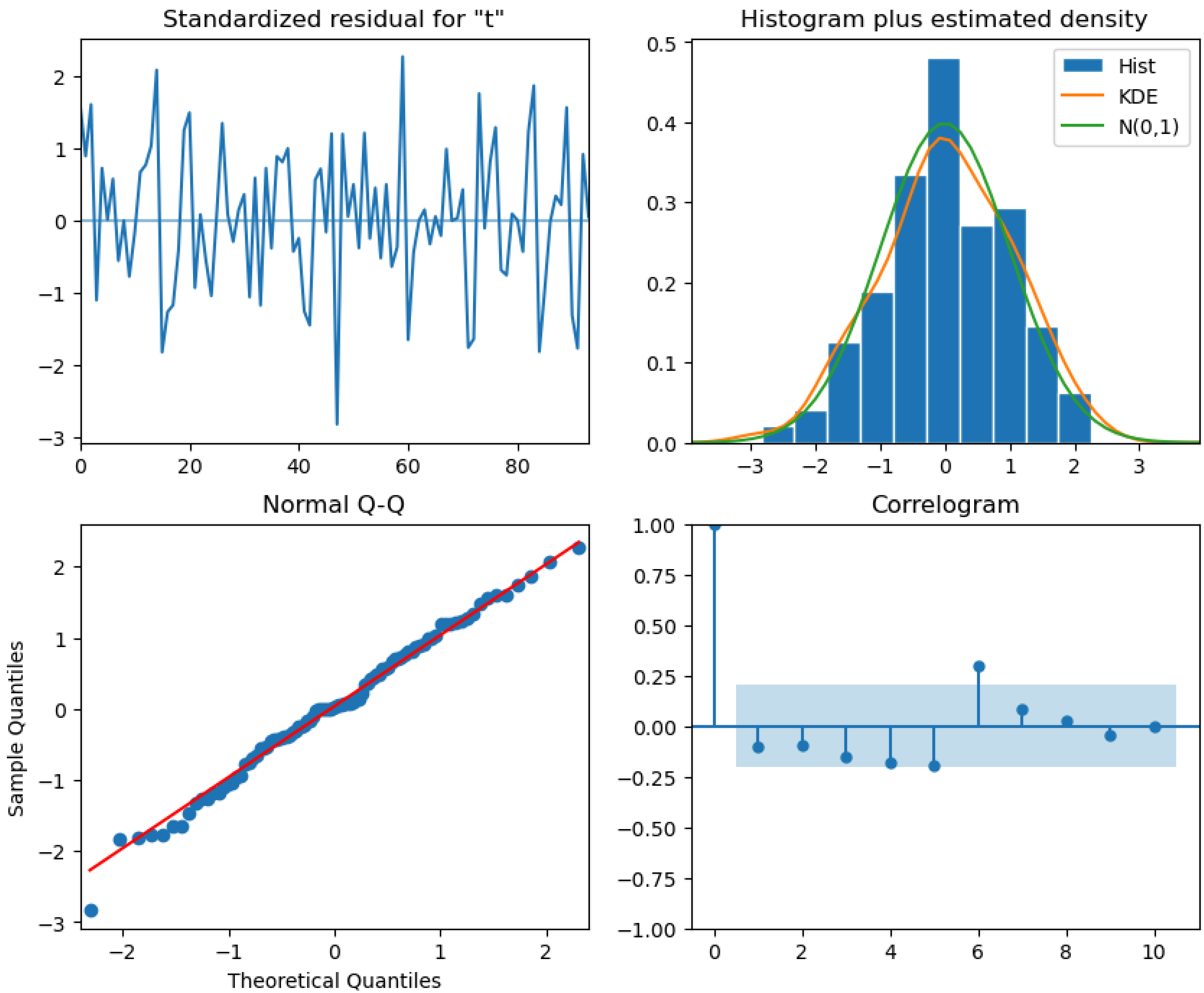
In [36]:
```
from pandas import datetime
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.arima.model import ARIMA
from matplotlib import pyplot
# line plot of residuals
residuals = pd.DataFrame(arima_fit.resid)
residuals.plot()
pyplot.show()
```

```
# density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
```





In [37]:
```
# 1. Model Diagnostics
arima_fit.plot_diagnostics(figsize=(10, 8))
plt.show()
```


Standardized residual for "t"
Histogram plus estimated density
Hist
KDE
N(0,1)
Normal Q-Q
Sample Quantiles
Theoretical Quantiles
Correlogram

In [38]:
```
# summary stats of residuals
print(residuals.describe())
```

```
               0
count  95.000000
mean    0.054663
std     2.475519
min    -6.756096
25%    -1.333934
50%     0.062304
75%     1.768500
max     5.403336
```

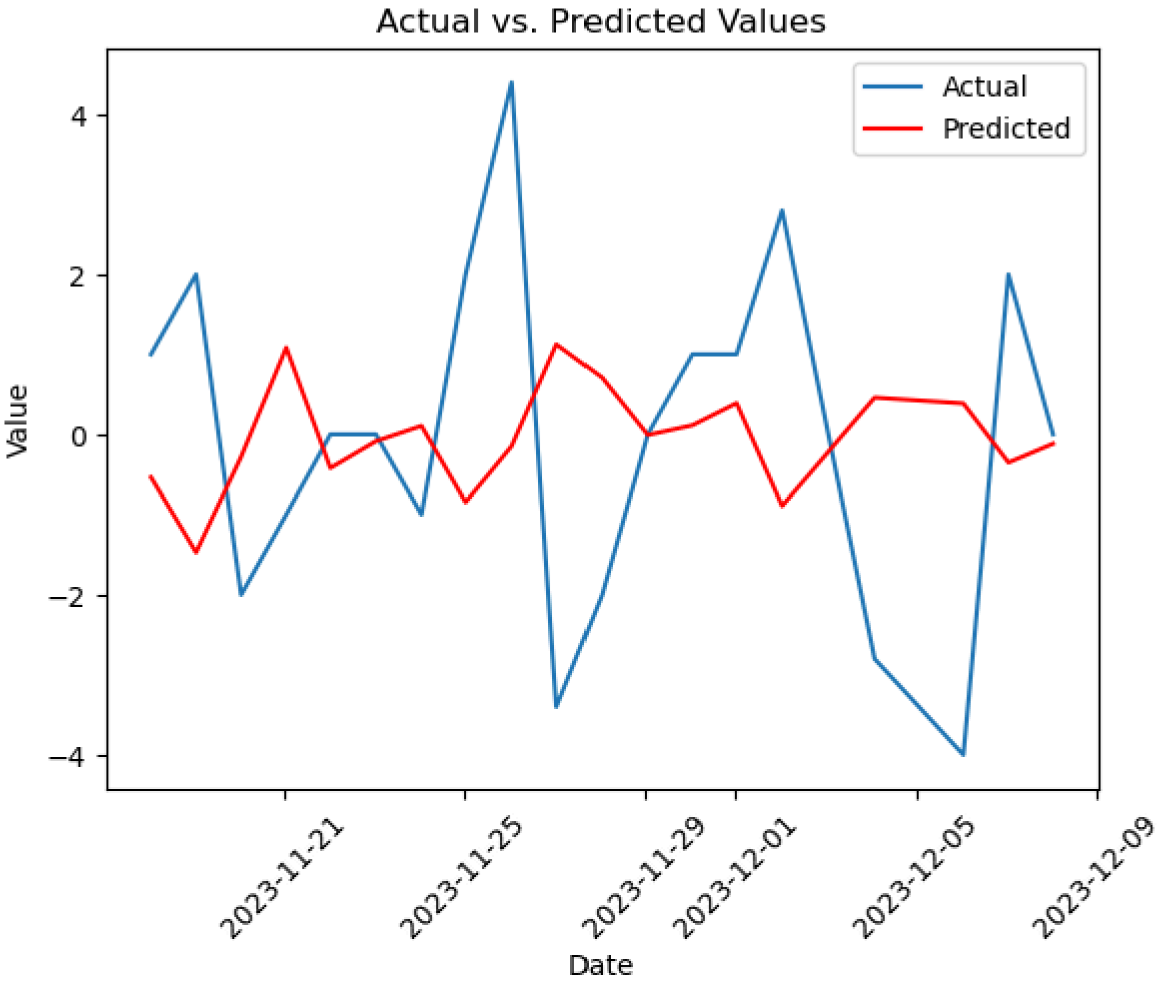
In [39]:
```
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# Assuming 'df' is your DataFrame containing the time series data

# Define the size of the training set
train_size = int(len(df) * 0.8)  # 80% of data for training

# Initialize lists to store forecasted values
history = list(df[:train_size]['temperature_diff'])
predictions = []

# Iterate through the test set
for t in range(train_size, len(df)):
    # Fit ARIMA model
    model = ARIMA(history, order=best_params)  # Replace (p, d, q) with appropriate values
    model_fit = model.fit()

    # Forecast next value
    forecast = model_fit.forecast()[0]

    # Print forecasted and actual values
    print(f"Forecast={forecast:.3f}, Actual={df.iloc[t]['temperature_diff']:.0f}")

    # Append forecast to predictions list
    predictions.append(forecast)

    # Update history with actual value from test set
    history.append(df.iloc[t]['temperature_diff'])
```

```
# Calculate and print RMSE
rmse = np.sqrt(mean_squared_error(df[train_size:]['temperature_diff'], predictions))
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
Forecast=-0.534, Actual=1
Forecast=-1.471, Actual=2
Forecast=-0.278, Actual=-2
Forecast=1.083, Actual=-1
Forecast=-0.419, Actual=0
Forecast=-0.085, Actual=0
Forecast=0.106, Actual=-1
Forecast=-0.845, Actual=2
Forecast=-0.146, Actual=4
Forecast=1.124, Actual=-3
Forecast=0.709, Actual=-2
Forecast=-0.008, Actual=0
Forecast=0.114, Actual=1
Forecast=0.390, Actual=1
Forecast=-0.892, Actual=3
Forecast=0.459, Actual=-3
Forecast=0.387, Actual=-4
Forecast=-0.352, Actual=2
Forecast=-0.117, Actual=0
Root Mean Squared Error (RMSE): 2.614496114308874
```

In [40]:

```
# Plot actual vs. predicted values
plt.plot(df[train_size:].index, df[train_size:]['temperature_diff'], label='Actual')
plt.plot(df[train_size:].index, predictions, color='red', label='Predicted')
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



In [45]:

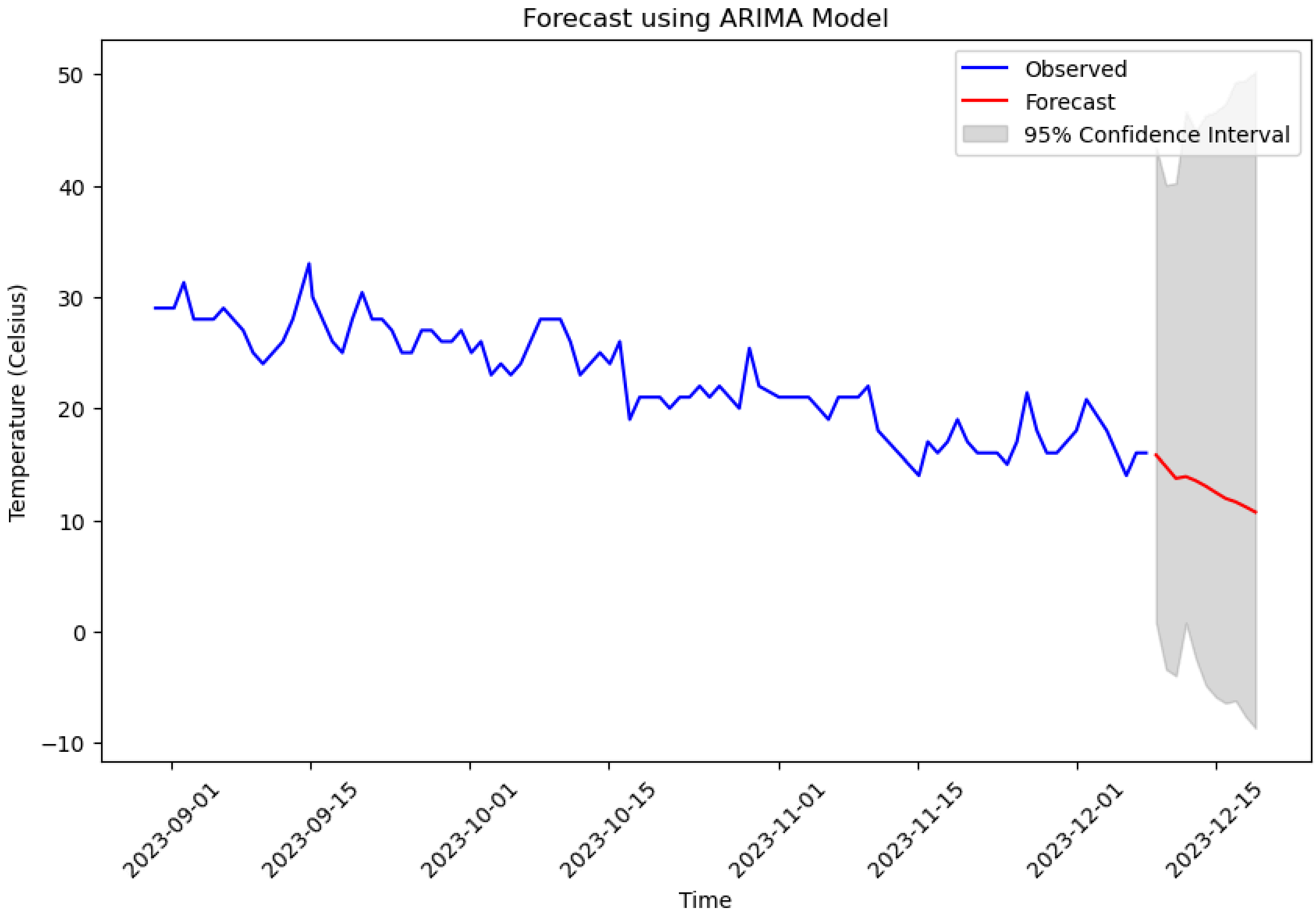
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model
model = ARIMA(df['temperature_diff'], order=best_params)
arima_fit = model.fit()

# Forecast future difference values
forecast_steps = 11  # Number of time steps to forecast into the future
```

```
forecast_diff = arima_fit.forecast(steps=forecast_steps)

# Convert forecasted difference values back to original scale
last_observed_value = df['temperature_celsius'].iloc[-1]  # Last observed value from the original series
forecast_original = np.cumsum(forecast_diff) + last_observed_value
print("forecast values :",forecast_original)
```

```
forecast values : 95     15.812848
96     14.799406
97     13.743167
98     13.908551
99     13.533627
100    13.050558
101    12.485067
102    11.945606
103    11.639395
104    11.195392
105    10.722270
Name: predicted_mean, dtype: float64
```

In [48]:

```
# Calculate mean and standard deviation of temperature data
mean_temp = df['temperature_celsius'].mean()
std_dev = df['temperature_celsius'].std()

# Convert confidence interval bounds to original scale
forecast = arima_fit.get_forecast(steps=forecast_steps)
forecast_conf_int = forecast.conf_int()
forecast_conf_int_original = forecast_conf_int * std_dev + mean_temp

# Plot forecasted values with 95% confidence interval in original scale
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['temperature_celsius'], label='Observed', color='blue')  # Plot observed values
plt.plot(pd.date_range(start=df.index[-1], periods=forecast_steps+1, freq='D')[1:], forecast_original, label='Forecast', color='r
plt.fill_between(pd.date_range(start=df.index[-1], periods=forecast_steps+1, freq='D')[1:], forecast_conf_int_original.iloc[:, 0]
plt.xlabel('Time')
plt.ylabel('Temperature (Celsius)')
plt.title('Forecast using ARIMA Model')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```


Forecast using ARIMA Model
Observed
Forecast
95% Confidence Interval
Temperature (Celsius)
50
40
30
20
10
0
−10
2023-09-01
2023-09-15
2023-10-01
2023-10-15
2023-11-01
2023-11-15
2023-12-01
2023-12-15
Time

```
In [ ]:
```