```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
warnings.filterwarnings("ignore")
```

```python
df = pd.read_csv('weatherHistory.csv', parse_dates=['Formatted Date'])
```

```python
print(df.head())
```

```
              Formatted Date        Summary Precip Type  Temperature (C)  \
0  2006-04-01 00:00:00+02:00  Partly Cloudy        rain         9.472222
1  2006-04-01 01:00:00+02:00  Partly Cloudy        rain         9.355556
2  2006-04-01 02:00:00+02:00  Mostly Cloudy        rain         9.377778
3  2006-04-01 03:00:00+02:00  Partly Cloudy        rain         8.288889
4  2006-04-01 04:00:00+02:00  Mostly Cloudy        rain         8.755556

   Apparent Temperature (C)  Humidity  Wind Speed (km/h)  \
0                  7.388889      0.89            14.1197
1                  7.227778      0.86            14.2646
2                  9.377778      0.89             3.9284
3                  5.944444      0.83            14.1036
4                  6.977778      0.83            11.0446

   Wind Bearing (degrees)  Visibility (km)  Loud Cover  Pressure (millibars)  \
0                   251.0          15.8263         0.0               1015.13
1                   259.0          15.8263         0.0               1015.63
2                   204.0          14.9569         0.0               1015.94
3                   269.0          15.8263         0.0               1016.41
4                   259.0          15.8263         0.0               1016.51

                       Daily Summary
0  Partly cloudy throughout the day.
1  Partly cloudy throughout the day.
2  Partly cloudy throughout the day.
3  Partly cloudy throughout the day.
4  Partly cloudy throughout the day.
```

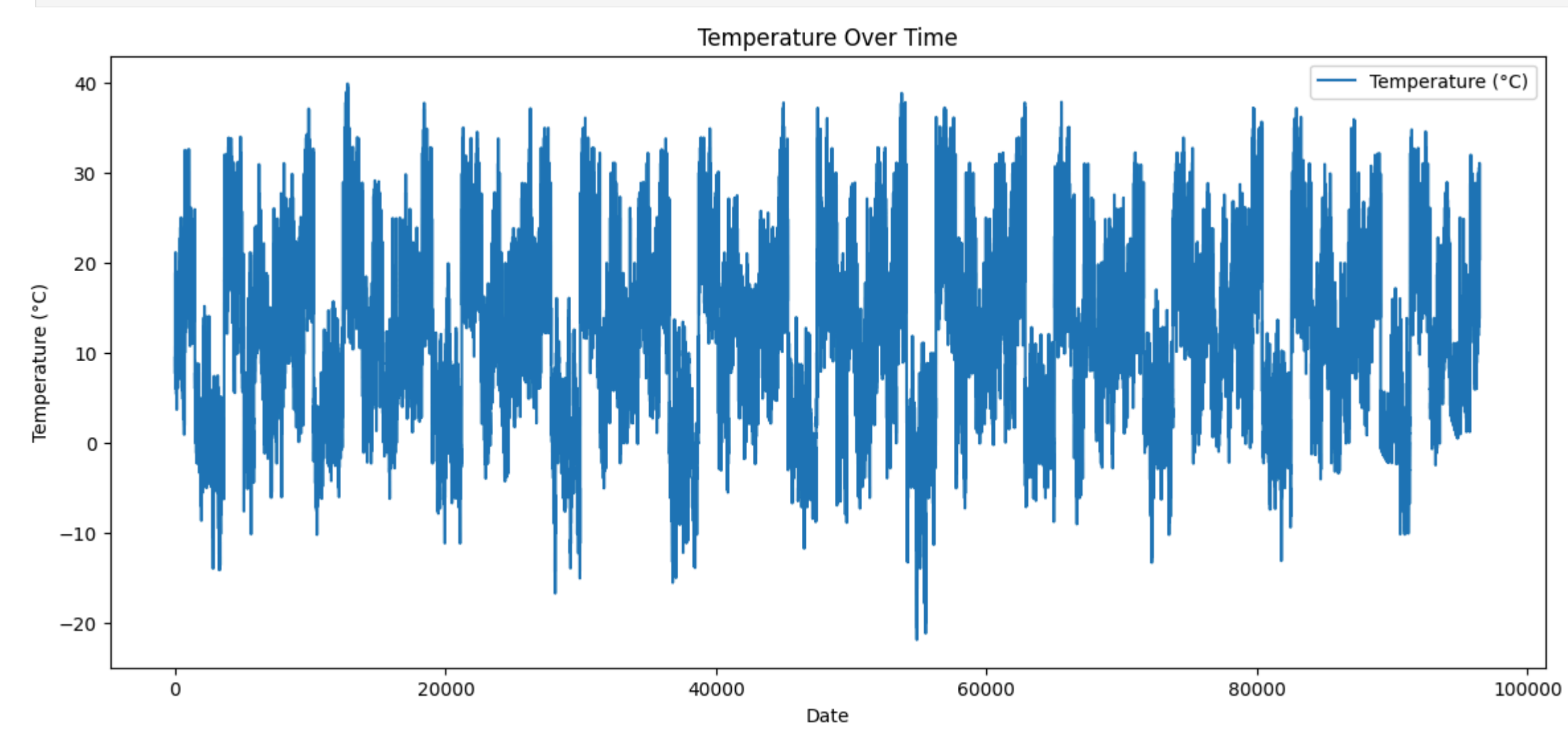
```python
# Display summary information including data types and non-null counts.
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Formatted Date            96453 non-null  object
 1   Summary                   96453 non-null  object
 2   Precip Type               95936 non-null  object
 3   Temperature (C)           96453 non-null  float64
 4   Apparent Temperature (C)  96453 non-null  float64
 5   Humidity                  96453 non-null  float64
 6   Wind Speed (km/h)         96453 non-null  float64
 7   Wind Bearing (degrees)    96453 non-null  float64
 8   Visibility (km)           96453 non-null  float64
 9   Loud Cover                96453 non-null  float64
 10  Pressure (millibars)      96453 non-null  float64
 11  Daily Summary             96453 non-null  object
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
None
```

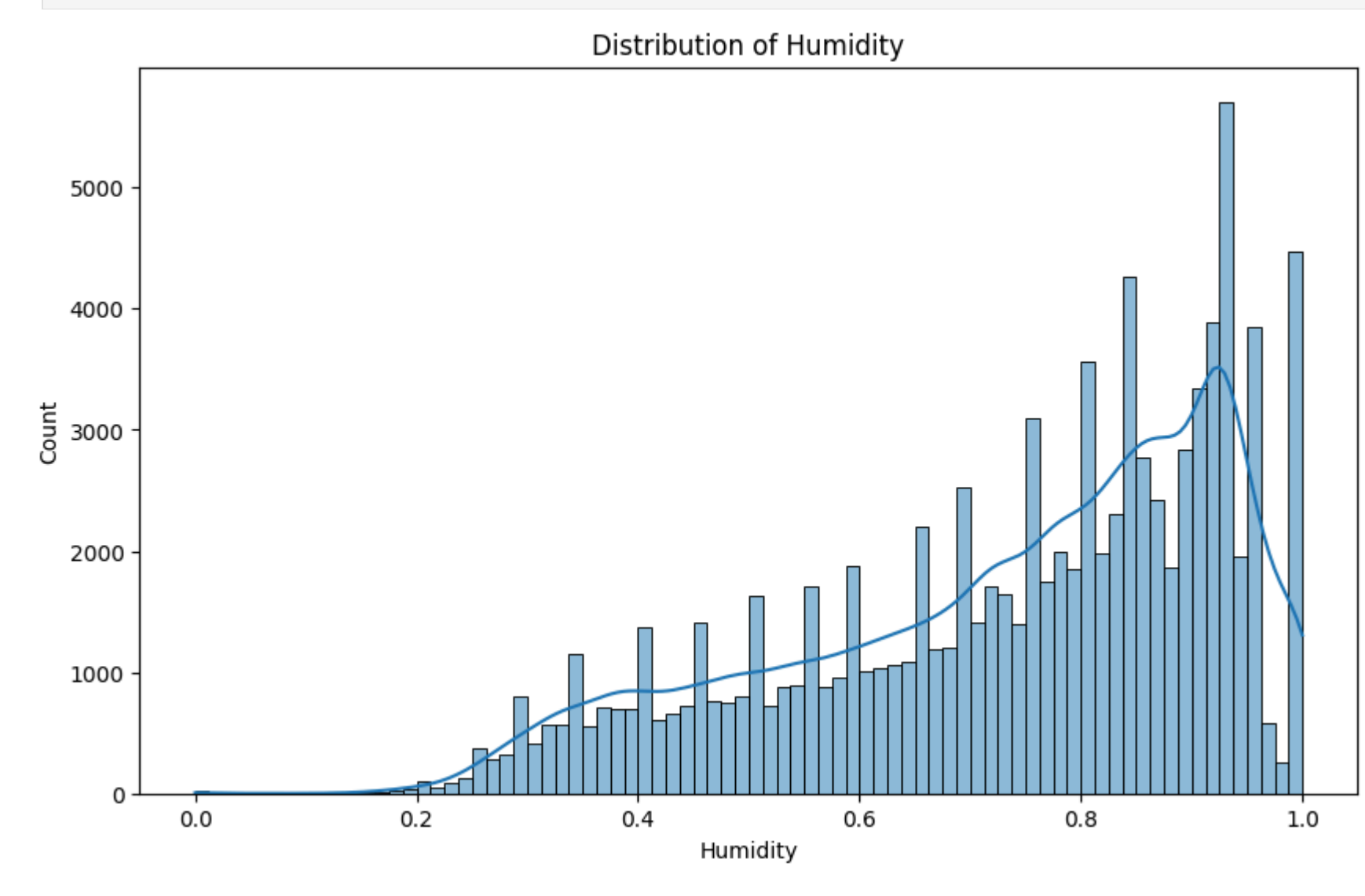
```python
# Show basic statistical details for numerical columns.
print(df.describe())
```

```
       Temperature (C)  Apparent Temperature (C)      Humidity  \
count     96453.000000              96453.000000  96453.000000
mean         11.932678                 10.855029      0.734899
std           9.551546                 10.696847      0.195473
min         -21.822222                -27.716667      0.000000
25%           4.688889                  2.311111      0.600000
50%          12.000000                 12.000000      0.780000
75%          18.838889                 18.838889      0.890000
max          39.905556                 39.344444      1.000000

       Wind Speed (km/h)  Wind Bearing (degrees)  Visibility (km)  Loud Cover  \
count       96453.000000            96453.000000     96453.000000     96453.0
mean           10.810640              187.509232        10.347325         0.0
std             6.913571              107.383428         4.192123         0.0
min             0.000000                0.000000         0.000000         0.0
25%             5.828200              116.000000         8.339800         0.0
50%             9.965900              180.000000        10.046400         0.0
75%            14.135800              290.000000        14.812000         0.0
max            63.852600              359.000000        16.100000         0.0

       Pressure (millibars)
count          96453.000000
mean            1003.235956
std              116.969906
min                0.000000
25%             1011.900000
50%             1016.450000
75%             1021.090000
max             1046.380000
```

```python
# Plot the temperature over time.
plt.figure(figsize=(14, 6))
plt.plot(df.index, df['Temperature (C)'], label='Temperature (°C)')
plt.title('Temperature Over Time')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.show()
```

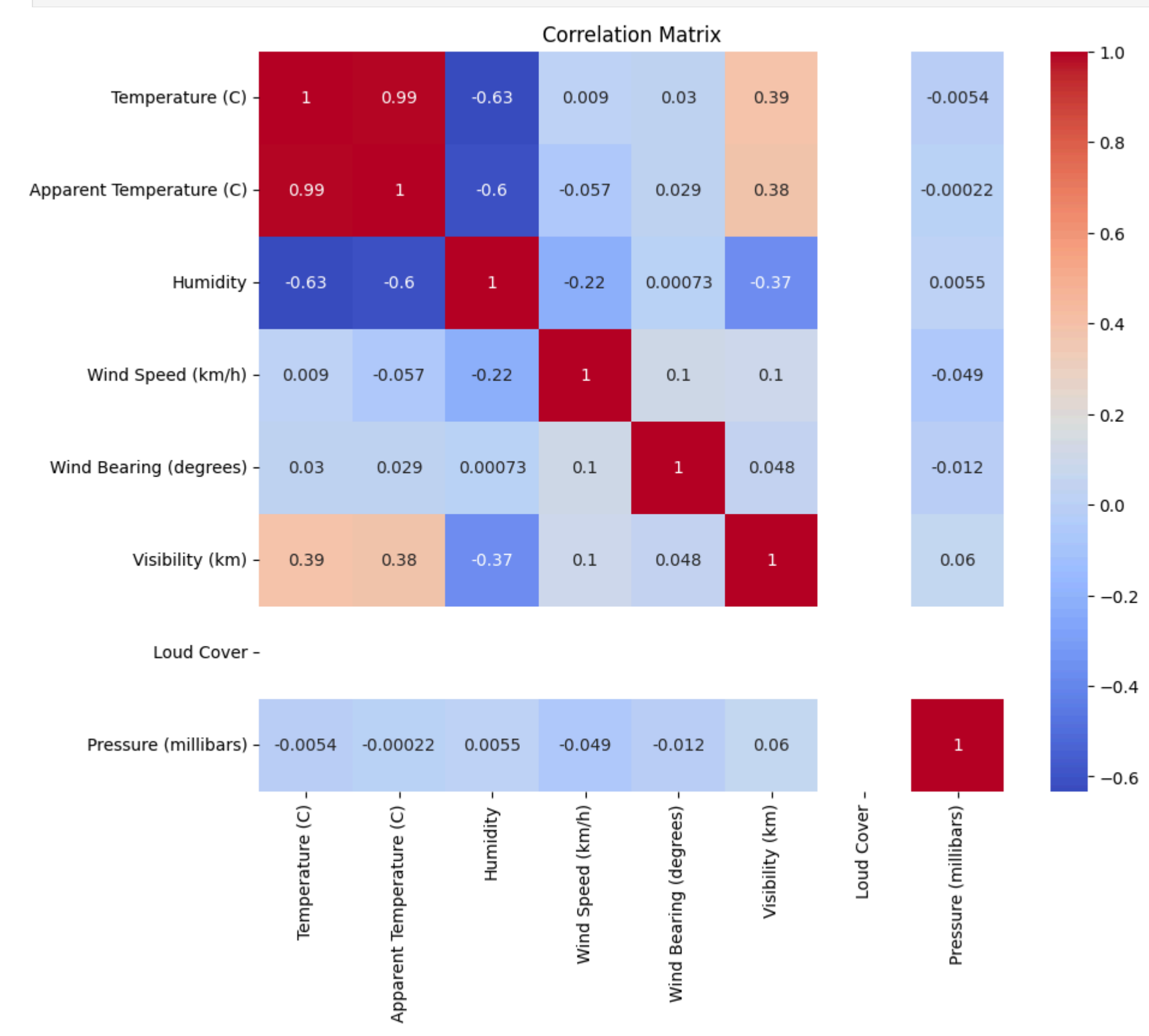



```python
# Plot a histogram of humidity with a kernel density estimate.
plt.figure(figsize=(10, 6))
sns.histplot(df['Humidity'], kde=True)
plt.title('Distribution of Humidity')
plt.xlabel('Humidity')
plt.show()
```

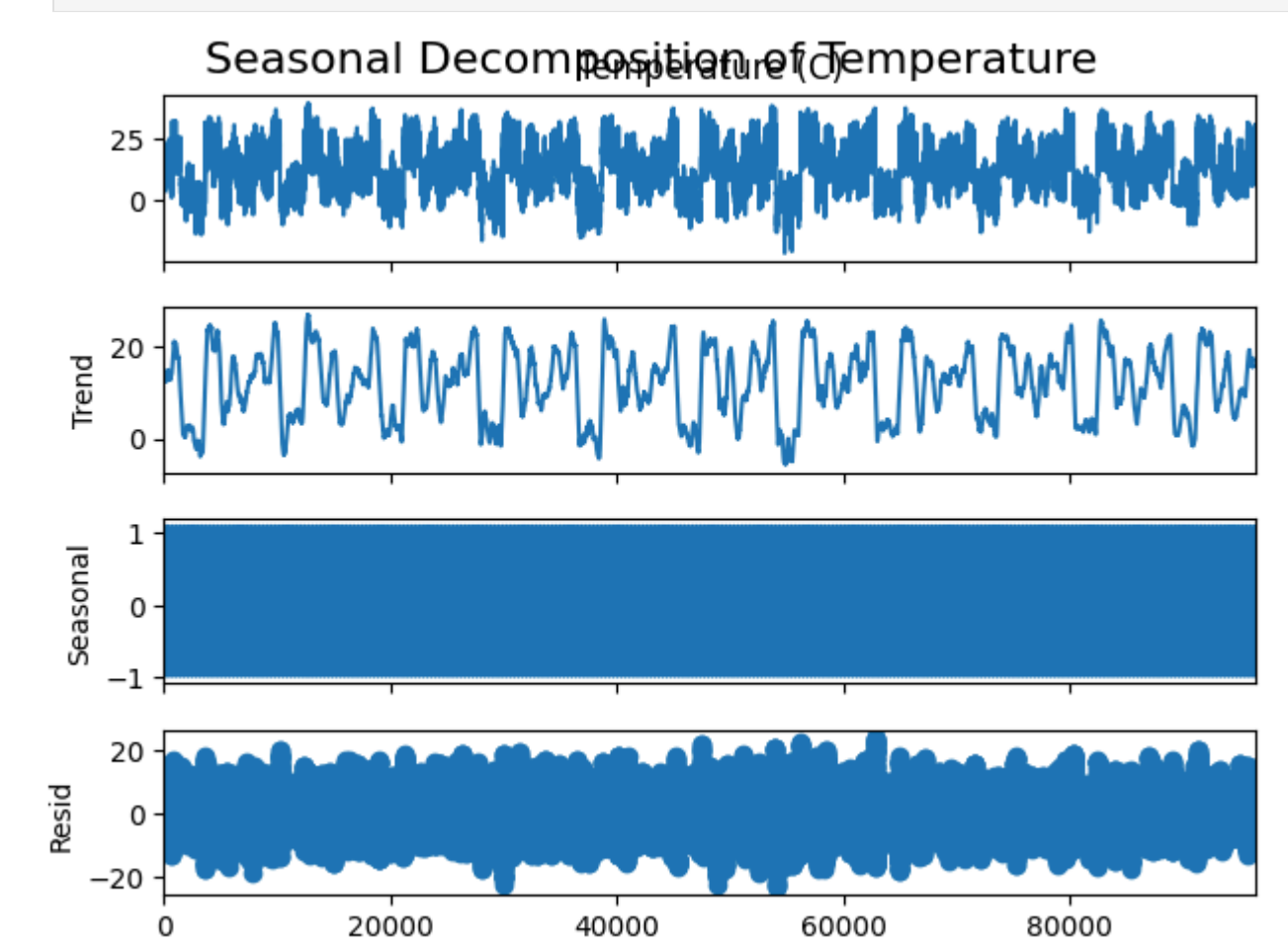



```python
df_numeric = df.drop(columns=['Precip Type', 'Summary', 'Daily Summary', 'Formatted Date'])
corr_matrix = df_numeric.corr()
```

```python
# Compute the correlation matrix for numerical features.
corr_matrix = df_numeric.corr()

# Plot the correlation matrix using a heatmap.
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

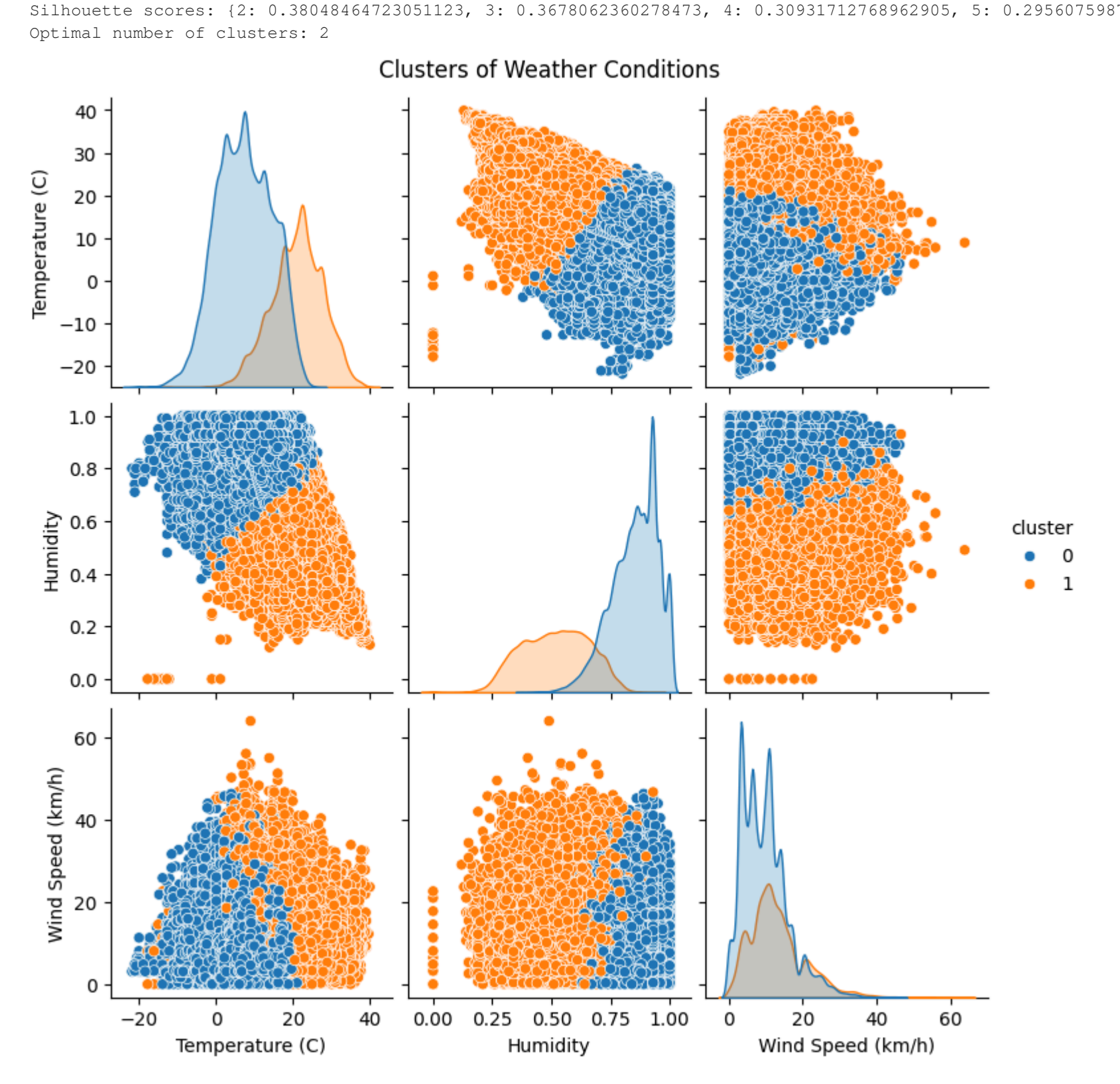



```python
# Decompose the temperature time series into trend, seasonal, and residual components.
decomposition = seasonal_decompose(df['Temperature (C)'].dropna(), model='additive', period=365)
decomposition.plot()
plt.suptitle('Seasonal Decomposition of Temperature', fontsize=16)
plt.show()
```

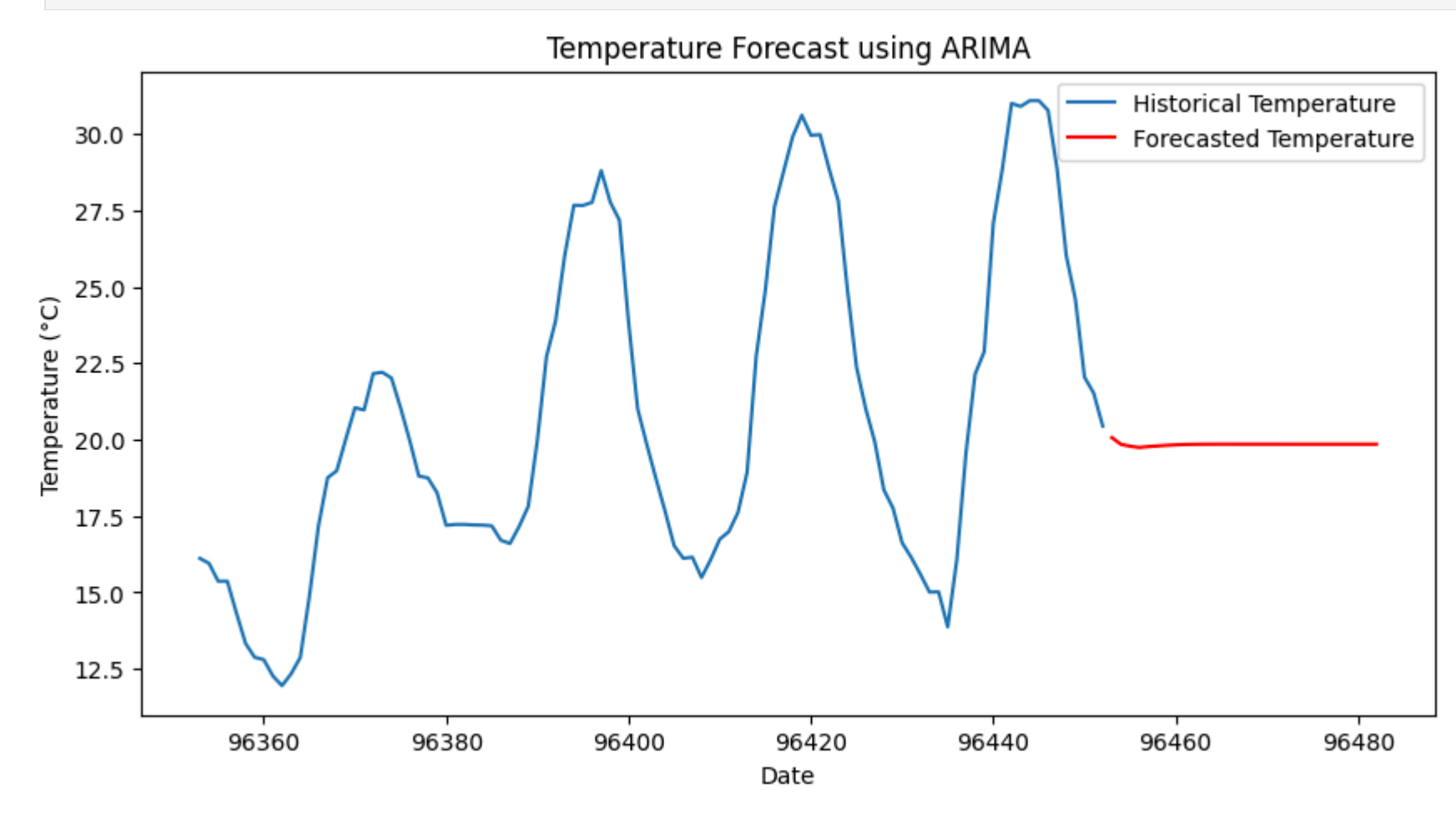



```python
# Select relevant features for clustering
features = df[['Temperature (C)', 'Humidity', 'Wind Speed (km/h)']].dropna()

# Standardize features to have a mean of 0 and standard deviation of 1.
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Find the optimal number of clusters using the silhouette score.
scores = {}
for k in range(2, 8):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(scaled_features)
    scores[k] = silhouette_score(scaled_features, labels)
print("Silhouette scores:", scores)

# Choose the optimal number of clusters (for example, highest silhouette score).
optimal_k = max(scores, key=scores.get)
print("Optimal number of clusters:", optimal_k)

# Fit KMeans using the optimal number and assign cluster labels.
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
features['cluster'] = kmeans.fit_predict(scaled_features)

# Visualize the clusters using a pairplot.
sns.pairplot(features, hue='cluster', diag_kind='kde')
plt.suptitle('Clusters of Weather Conditions', y=1.02)
plt.show()
```

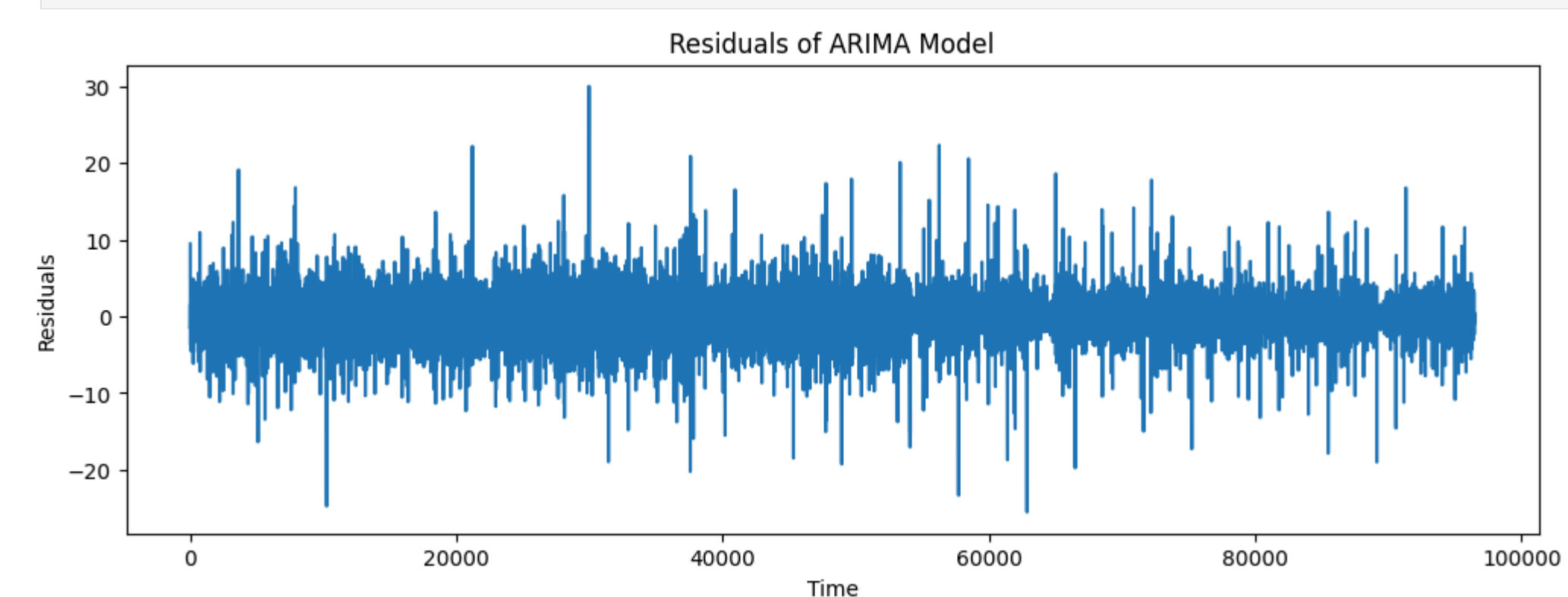
```
Silhouette scores: {2: 0.38048464723051123, 3: 0.3678062360278473, 4: 0.30931712768962905, 5: 0.29560759873258036, 6: 0.2725170481556916, 7: 0.2750216562336027}
Optimal number of clusters: 2
```

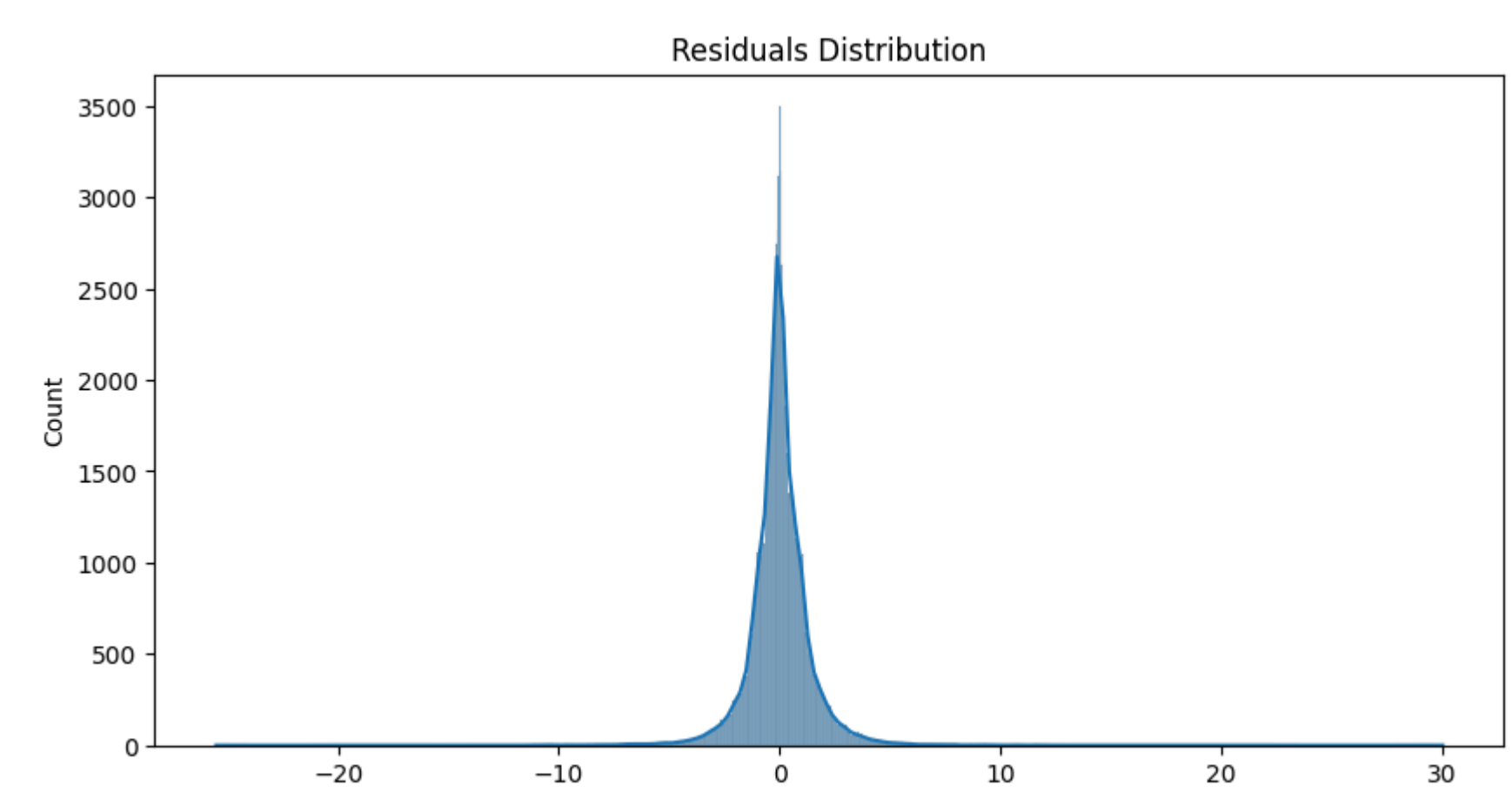



```python
# Fit an ARIMA model on temperature data. Drop missing values for a clean time series.
temperature_series = df['Temperature (C)'].dropna()

# Create and fit the ARIMA model with parameters (p=5, d=1, q=0)
model = ARIMA(temperature_series, order=(5, 1, 0))
model_fit = model.fit()

# Forecast the next 30 time steps.
forecast = model_fit.forecast(steps=30)

# Plot historical data and the forecast.
plt.figure(figsize=(10, 5))
plt.plot(temperature_series.tail(100), label='Historical Temperature')
plt.plot(forecast, label='Forecasted Temperature', color='red')
plt.title('Temperature Forecast using ARIMA')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.show()
```



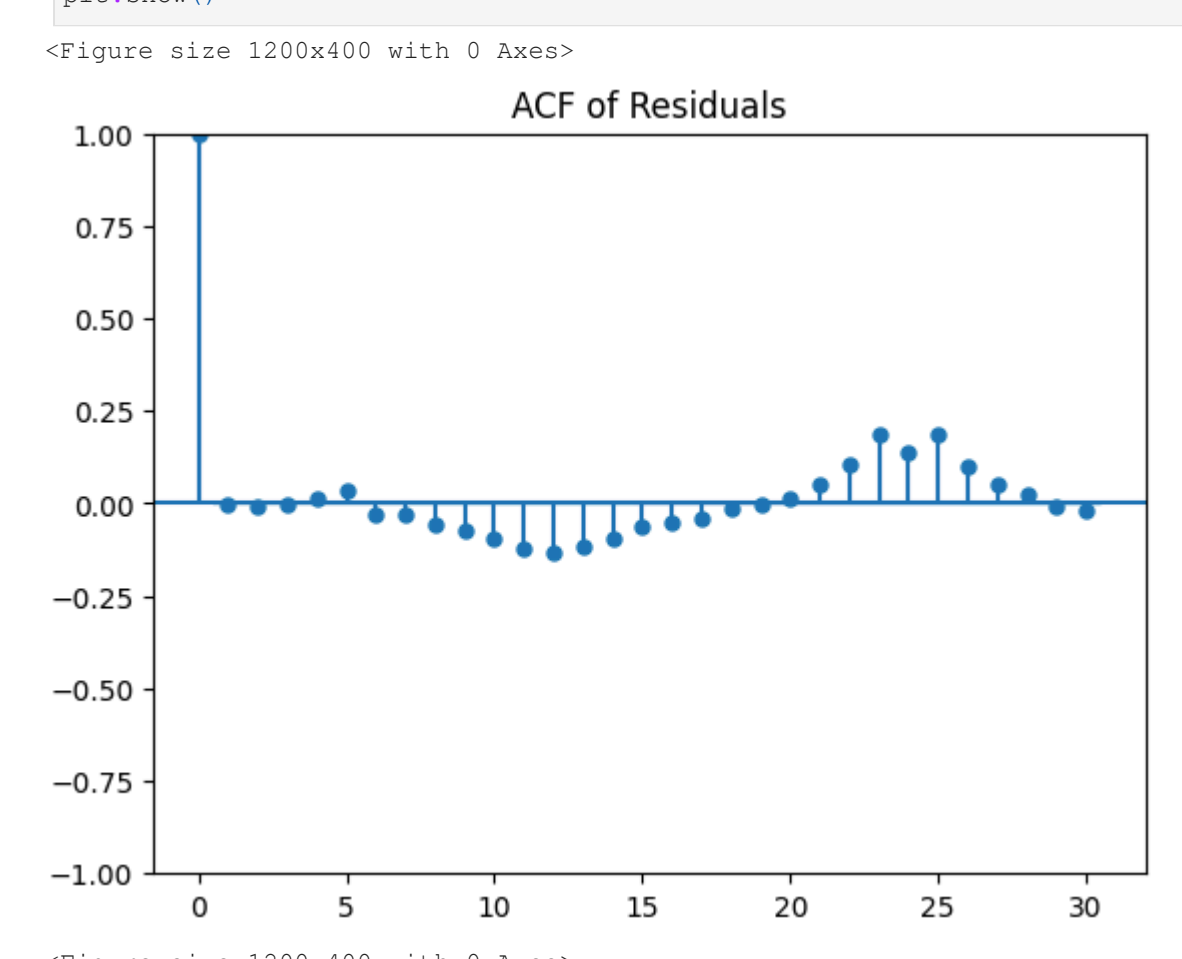

```python
# Extract residuals
residuals = model_fit.resid

# Plot the residuals over time
plt.figure(figsize=(12, 4))
plt.plot(residuals)
plt.title('Residuals of ARIMA Model')
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.show()

# Plot the histogram and KDE of the residuals
plt.figure(figsize=(10, 5))
sns.histplot(residuals, kde=True)
plt.title('Residuals Distribution')
plt.xlabel('Residual')
plt.show()
```



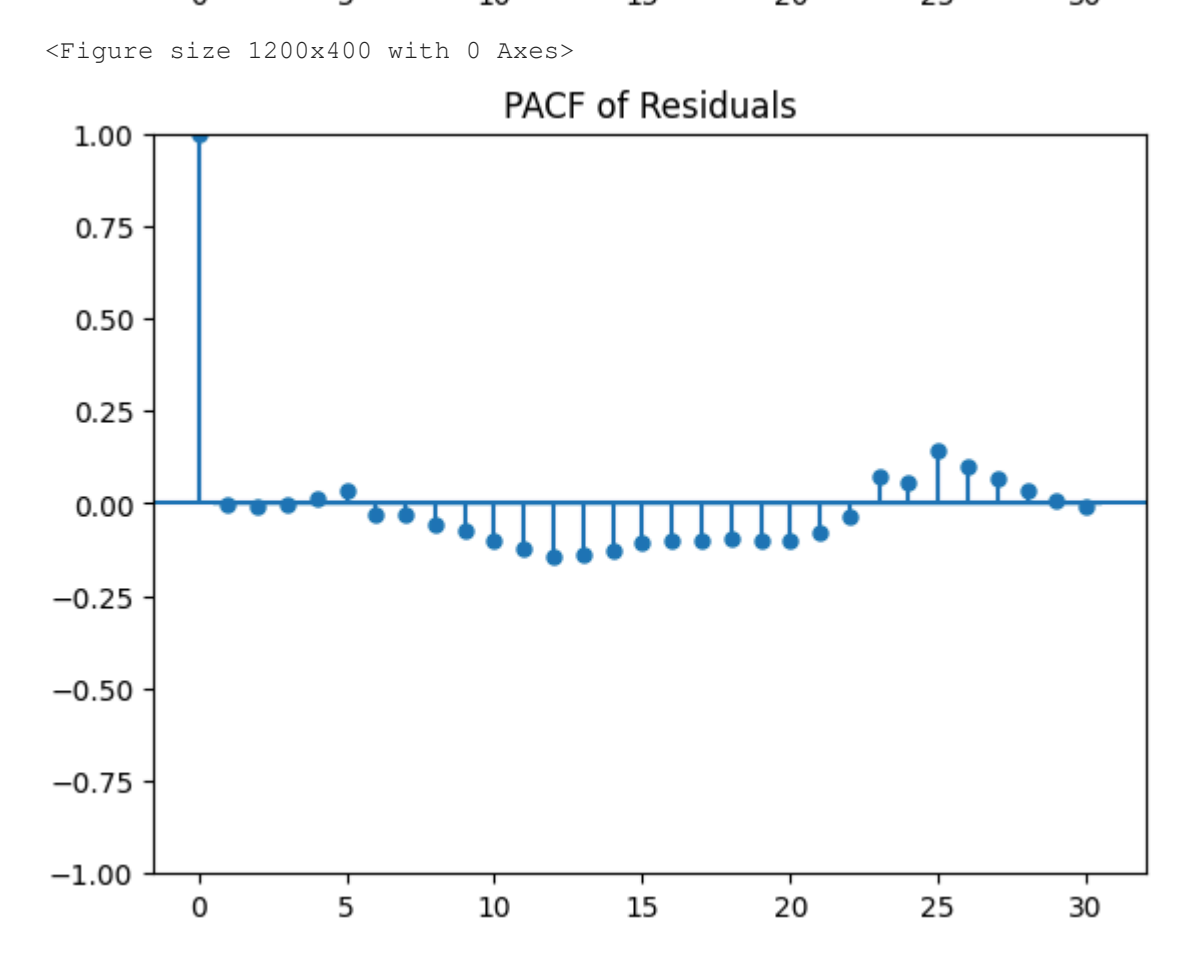



```python
plt.figure(figsize=(12, 4))
plot_acf(residuals, lags=30)
plt.title('ACF of Residuals')
plt.show()

plt.figure(figsize=(12, 4))
plot_pacf(residuals, lags=30, method='ywm')
plt.title('PACF of Residuals')
plt.show()
```

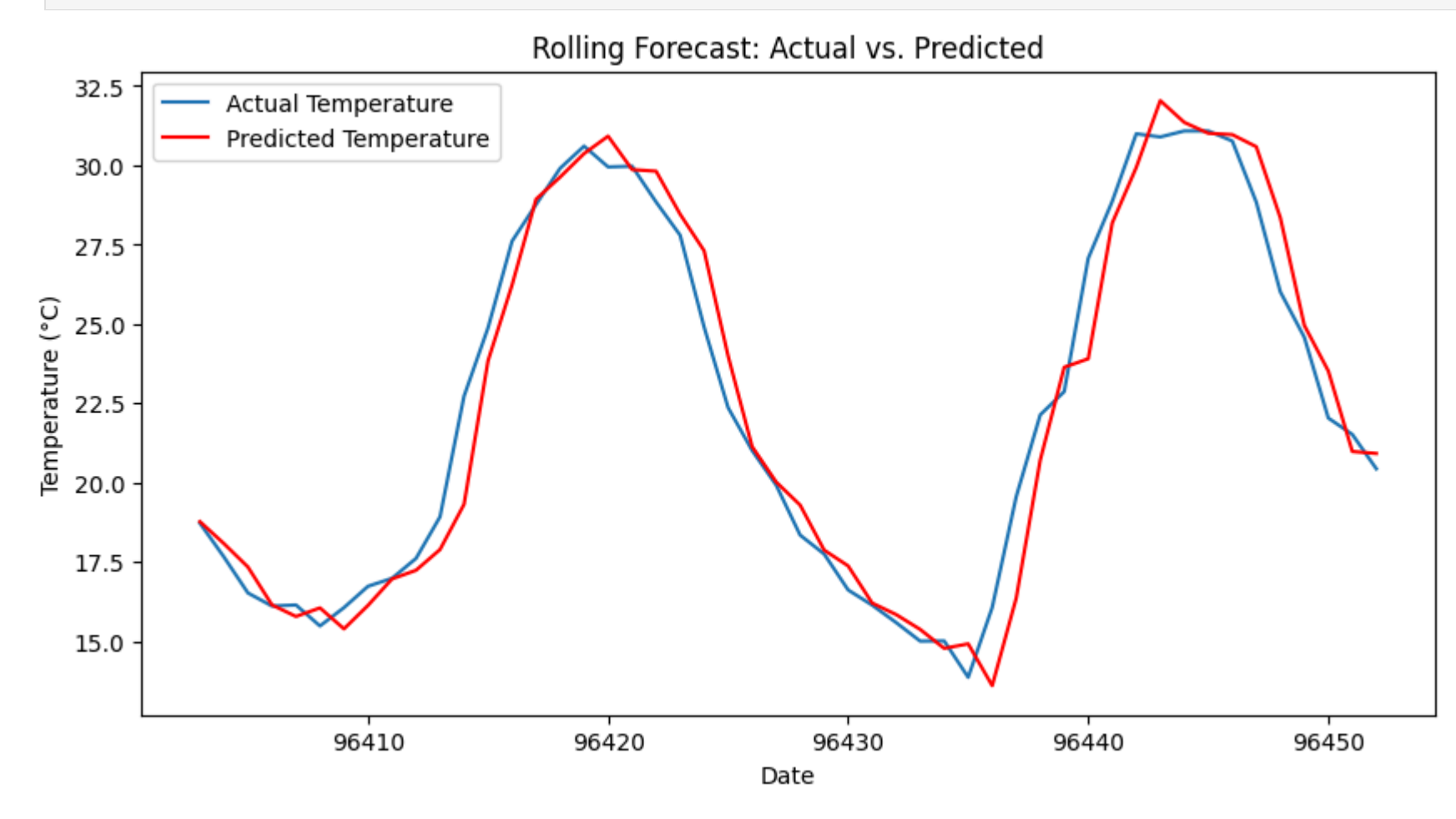
```
<Figure size 1200x400 with 0 Axes>
```




```
<Figure size 1200x400 with 0 Axes>
```




```python
# last 50 observations as our test set.
train = temperature_series[:-50]
test = temperature_series[-50:]
predictions = []

# Rolling forecast
history = list(train)
for t in range(len(test)):
    model = ARIMA(history, order=(5, 1, 0))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=1)
    predictions.append(forecast[0])
    history.append(test.iloc[t])

# Plot the test data and predictions
plt.figure(figsize=(10, 5))
plt.plot(test.index, test, label='Actual Temperature')
plt.plot(test.index, predictions, label='Predicted Temperature', color='red')
plt.title('Rolling Forecast: Actual vs. Predicted')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.show()
```




```python
# Calculate forecast accuracy metrics
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(test, predictions)
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 0.8726484543969519
```

In [ ]: