```
import pandas as pd
import os
import pathlib
from google.colab import drive
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from prophet import Prophet
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.arima.model import ARIMA


#to mound the google drive to colab
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#reteriving data from google drive
folder_path = '/content/drive/MyDrive/Test/Test_Data_Files'

file_list = os.listdir(folder_path)
dfs = []
for file in file_list:
    if file.endswith('.txt'):
        file_path = os.path.join(folder_path, file)
        df = pd.read_csv(file_path, skiprows=3, delimiter='\s+', encoding='utf-8', header=None)
        with open(file_path, 'r', encoding='latin1') as f:
            first_line = f.readline().strip()
        date_value = first_line.split()[0]
        df['Date'] = pd.to_datetime(date_value, format='%d/%m/%y')
        dfs.append(df)


result_df = pd.concat(dfs, ignore_index=True)

print(result_df)
```

```
                0      1     2     3     4     5     6     7       8    9  10  \
     0         0:00  254.0   3.2   3.2  40.0  85.0  24.7  14.1  1014.1  0.0   0
     1         0:01  254.0   0.0   3.2  40.0  86.0  24.7  14.1  1014.0  0.0   0
     2         0:02  254.0   3.2   3.2  40.0  86.0  24.6  14.1  1014.1  0.0   0
     3         0:03  254.0   0.0   3.2  40.0  86.0  24.6  14.1  1014.1  0.0   0
     4         0:04  254.0   0.0   1.6  40.0  86.0  24.6  14.1  1014.1  0.0   0
     ...        ...    ...   ...   ...   ...   ...   ...   ...     ...  ...  ..
     1629530  23:55  163.0  11.3  14.5  56.0  79.0  15.4  11.9  1014.4  0.0   0
     1629531  23:56  161.0   8.0  14.5  56.0  79.0  15.4  11.9  1014.3  0.0   0
     1629532  23:57  164.0   6.4  11.3  56.0  79.0  15.4  11.9  1014.3  0.0   0
     1629533  23:58  201.0   6.4  12.9  56.0  79.0  15.4  11.9  1014.3  0.0   0
     1629534  23:59  176.0  11.3  11.3  57.0  79.0  15.4  11.9  1014.3  0.0   0

               11   12   13       Date
     0        11.6  0.0  0.0 2021-06-04
     1        11.8  0.0  0.0 2021-06-04
     2        11.8  0.0  0.0 2021-06-04
     3        11.8  0.0  0.0 2021-06-04
     4        11.8  0.0  0.0 2021-06-04
     ...       ...  ...  ...        ...
     1629530   8.4  0.0  0.0 2021-02-24
     1629531   8.4  0.0  0.0 2021-02-24
     1629532   8.4  0.0  0.0 2021-02-24
     1629533   8.4  0.0  0.0 2021-02-24
     1629534   8.4  0.0  0.0 2021-02-24

     [1629535 rows x 15 columns]
```

```
print(result_df.Date.min(), result_df.Date.max())
```

```
     2021-01-01 00:00:00 2024-02-06 00:00:00
```

```
#adding columnnames to the dataframe
with open('/content/drive/MyDrive/Test/column_names.txt', 'r') as file:
    column_names = file.readline().strip().split()

result_df.columns = column_names
```

```
total_records = result_df.shape[0]
print("Total number of records:", total_records)
```

```
Total number of records: 1629535
```

```
result_df.tail()
```

	Time	Wind_Dir	Wind_Spd	Wind_Gust	Hum_In	Humidity	Temp_In	Temp	Raw_Barom	UV	Solar	Dew_Point	DailyRain	RainRate
1629530	23:55	163.0	11.3	14.5	56.0	79.0	15.4	11.9	1014.4	0.0	0	8.4	0.0	0.0
1629531	23:56	161.0	8.0	14.5	56.0	79.0	15.4	11.9	1014.3	0.0	0	8.4	0.0	0.0
1629532	23:57	164.0	6.4	11.3	56.0	79.0	15.4	11.9	1014.3	0.0	0	8.4	0.0	0.0

```
result_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1629535 entries, 0 to 1629534
Data columns (total 15 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   Time       1629535 non-null  object
 1   Wind_Dir   1629535 non-null  float64
 2   Wind_Spd   1629535 non-null  float64
 3   Wind_Gust  1629535 non-null  float64
 4   Hum_In     1629535 non-null  float64
 5   Humidity   1629535 non-null  float64
 6   Temp_In    1629535 non-null  float64
 7   Temp       1629535 non-null  float64
 8   Raw_Barom  1629535 non-null  float64
 9   UV         1629535 non-null  float64
 10  Solar      1629535 non-null  int64
 11  Dew_Point  1629535 non-null  float64
 12  DailyRain  1629535 non-null  float64
 13  RainRate   1629535 non-null  float64
 14  Date       1629535 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(12), int64(1), object(1)
memory usage: 186.5+ MB
```

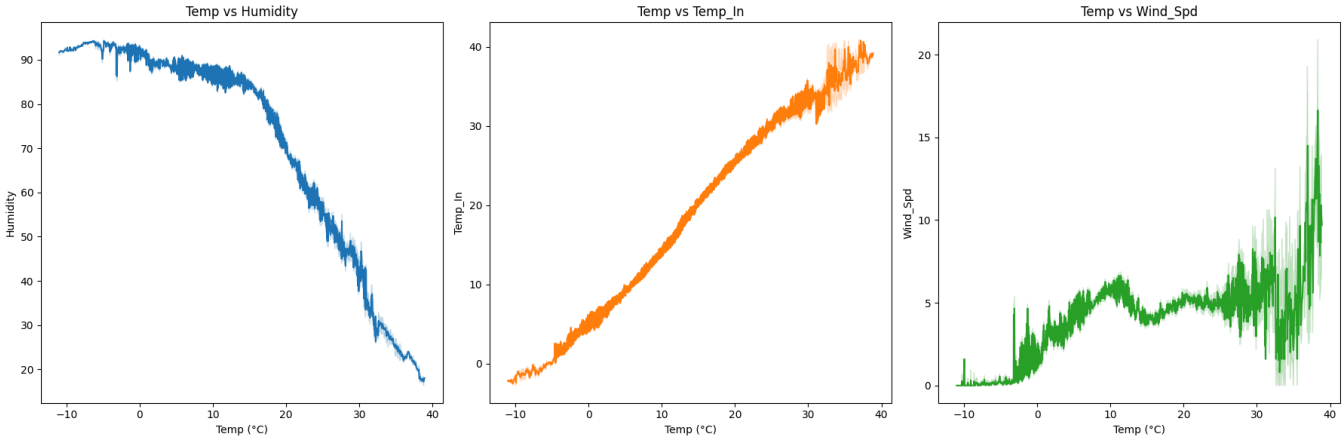
```
result_df.describe()
```

	Wind_Dir	Wind_Spd	Wind_Gust	Hum_In	Humidity	Temp_In	Temp	Raw_Barom	UV	
count	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1.629535e+06	1
mean	2.012897e+02	4.630836e+00	8.651249e+00	5.962888e+01	8.371515e+01	1.566787e+01	1.098527e+01	1.008632e+03	5.206977e-01	1
min	0.000000e+00	0.000000e+00	0.000000e+00	2.400000e+01	1.600000e+01	-2.780000e+00	-1.100000e+01	9.497200e+02	0.000000e+00	0
25%	1.360000e+02	0.000000e+00	3.200000e+00	5.100000e+01	7.800000e+01	1.044000e+01	6.700000e+00	1.001900e+03	0.000000e+00	0
50%	2.190000e+02	3.200000e+00	8.000000e+00	5.800000e+01	8.800000e+01	1.506000e+01	1.080000e+01	1.009600e+03	0.000000e+00	0

```
result_df.isnull().any()
```

```
Time         False
Wind_Dir     False
Wind_Spd     False
Wind_Gust    False
Hum_In       False
Humidity     False
Temp_In      False
Temp         False
Raw_Barom    False
UV           False
Solar        False
Dew_Point    False
DailyRain    False
RainRate     False
Date         False
dtype: bool
```

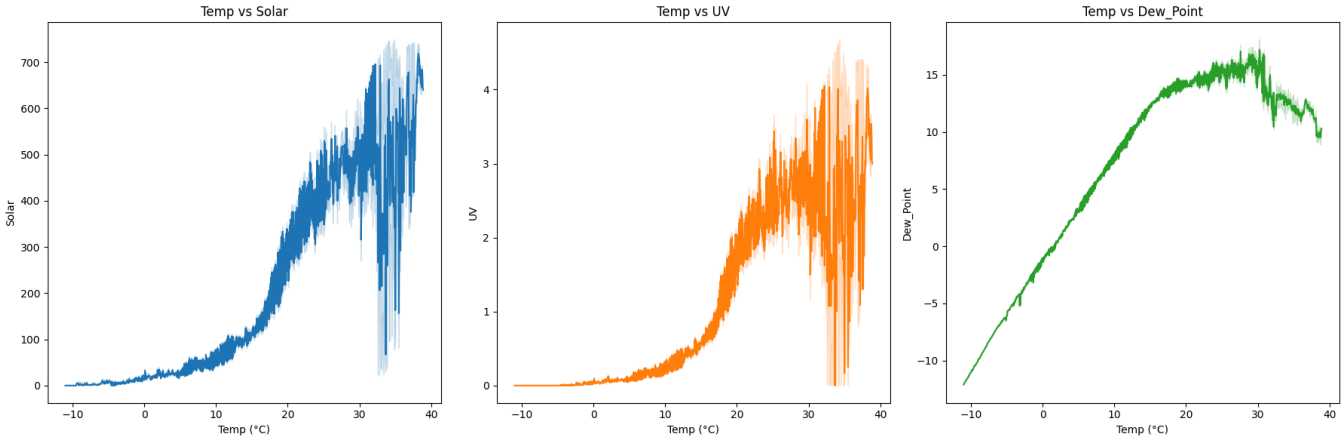
```
#plot to analyse temprature with humidity, Temp In, and WindSpeed
import matplotlib.pyplot as plt
import seaborn as sns

x_color = '#1f77b4'
y_color = '#ff7f0e'
z_color = '#2ca02c'


variable_pairs = [
    (result_df['Temp'], result_df['Humidity']),
    (result_df['Temp'], result_df['Temp_In']),
    (result_df['Temp'], result_df['Wind_Spd']),
]


fig, axes = plt.subplots(1, 3, figsize=(18, 6))


for i, (x, y) in enumerate(variable_pairs):
    color = x_color if i == 0 else (y_color if i == 1 else z_color)
    sns.lineplot(data=result_df, x=x, y=y, ax=axes[i], color=color)
    axes[i].set_title(f"{x.name} vs {y.name}")
    axes[i].set_xlabel(x.name+' (°C)')
    axes[i].set_ylabel(y.name)


plt.tight_layout()
plt.show()
```



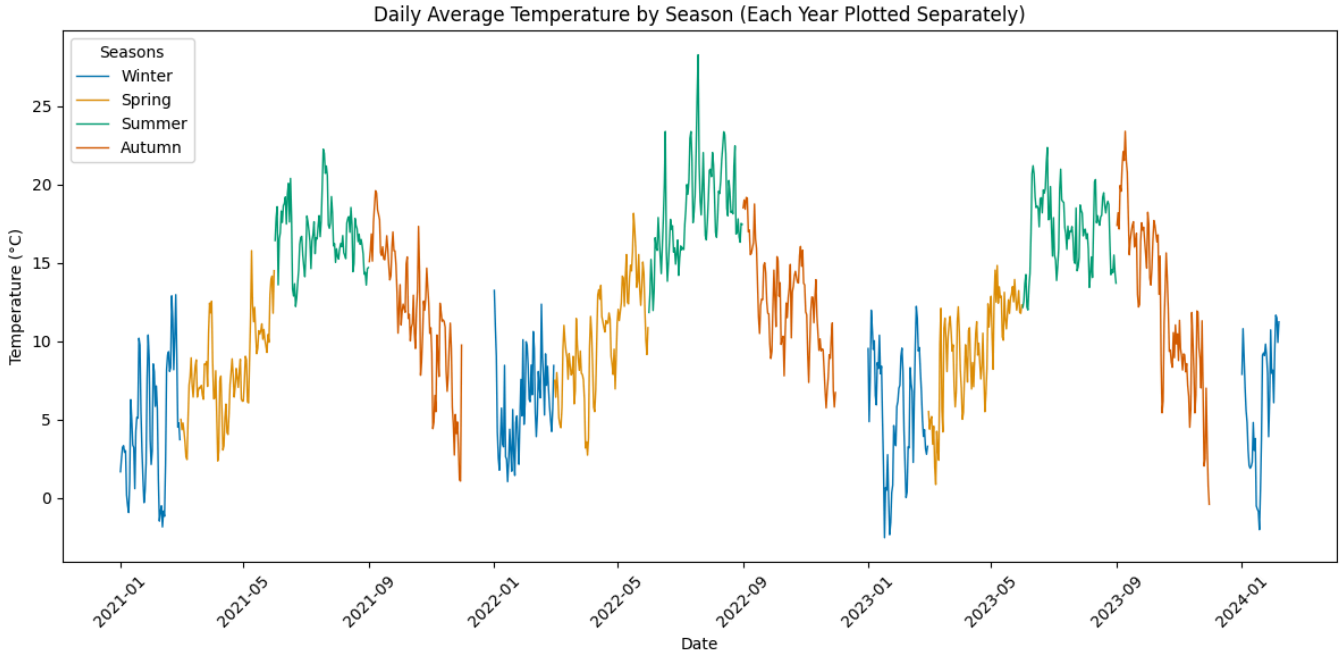

```
#correlation between temprature, humidity, Temp In, and WindSpeed
variable_pairs = [
    ('Temp', 'Humidity'),
    ('Temp_In', 'Temp'),
    ('Temp', 'Wind_Spd'),
]


correlation_coefficients = {}

for pair in variable_pairs:
    x, y = pair
    correlation_coefficients[pair] = result_df[x].corr(result_df[y])


for pair, correlation in correlation_coefficients.items():
    print(f"Correlation between {pair[0]} and {pair[1]}: {correlation:.2f}")
```

```
Correlation between Temp and Humidity: -0.54
Correlation between Temp_In and Temp: 0.94
Correlation between Temp and Wind_Spd: 0.10
```

```
#plot to analyse temprature with Solar, UV, and DewPoint
import matplotlib.pyplot as plt
import seaborn as sns

x_color = '#1f77b4'
y_color = '#ff7f0e'
z_color = '#2ca02c'


variable_pairs = [
    (result_df['Temp'], result_df['Solar']),
    (result_df['Temp'], result_df['UV']),
    (result_df['Temp'], result_df['Dew_Point']),
]


fig, axes = plt.subplots(1, 3, figsize=(18, 6))


for i, (x, y) in enumerate(variable_pairs):
    color = x_color if i == 0 else (y_color if i == 1 else z_color)
    sns.lineplot(data=result_df, x=x, y=y, ax=axes[i], color=color)
    axes[i].set_title(f"{x.name} vs {y.name}")
    axes[i].set_xlabel(x.name+' (°C)')
    axes[i].set_ylabel(y.name)


plt.tight_layout()
plt.show()
```



```
#correlation between temprature, Solar, UV, and DewPoint
variable_pairs = [
    ('Temp', 'Solar'),
    ('Temp', 'UV'),
    ('Temp', 'Dew_Point'),
]


correlation_coefficients = {}

for pair in variable_pairs:
    x, y = pair
    correlation_coefficients[pair] = result_df[x].corr(result_df[y])


for pair, correlation in correlation_coefficients.items():
    print(f"Correlation between {pair[0]} and {pair[1]}: {correlation:.2f}")

 Correlation between Temp and Solar: 0.48
 Correlation between Temp and UV: 0.49
 Correlation between Temp and Dew_Point: 0.90


#Grpah shows seasonal wise temperature
result_df['Date'] = pd.to_datetime(result_df['Date'])

result_df.set_index('Date', inplace=True)

# Calculate daily average temperature
daily_avg_temp = result_df['Temp'].resample('D').mean()

# Convert daily_avg_temp to a DataFrame with 'Date' as a column
daily_avg_temp_df = daily_avg_temp.to_frame()

daily_avg_temp_df.reset_index(inplace=True)

# Define the seasons based on the month
seasons = ['Winter', 'Spring', 'Summer', 'Autumn']
season_bins = [0, 3, 6, 9, 12]  # January-March, April-June, July-September, October-December
daily_avg_temp_df['Season'] = pd.cut(daily_avg_temp_df['Date'].dt.month, bins=season_bins, labels=seasons, right=False)

# Plot daily average temperature by season for each year separately
plt.figure(figsize=(12, 6), dpi=100)
for year in daily_avg_temp_df['Date'].dt.year.unique():
    year_data = daily_avg_temp_df[daily_avg_temp_df['Date'].dt.year == year]
    sns.lineplot(data=year_data, x='Date', y='Temp', hue='Season', palette='colorblind', linewidth=1)

handles, labels = plt.gca().get_legend_handles_labels()
season_handles = handles[:4]
season_labels = labels[:4]
plt.legend(season_handles, season_labels, title='Seasons', loc='upper left')

plt.title('Daily Average Temperature by Season (Each Year Plotted Separately)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
print(result_df.index)
print(result_df.columns)
```

```
DatetimeIndex(['2021-06-04', '2021-06-04', '2021-06-04', '2021-06-04',
               '2021-06-04', '2021-06-04', '2021-06-04', '2021-06-04',
               '2021-06-04', '2021-06-04',
               ...
               '2021-02-24', '2021-02-24', '2021-02-24', '2021-02-24',
               '2021-02-24', '2021-02-24', '2021-02-24', '2021-02-24',
               '2021-02-24', '2021-02-24'],
              dtype='datetime64[ns]', name='Date', length=1629535, freq=None)
Index(['Time', 'Wind_Dir', 'Wind_Spd', 'Wind_Gust', 'Hum_In', 'Humidity',
       'Temp_In', 'Temp', 'Raw_Barom', 'UV', 'Solar', 'Dew_Point', 'DailyRain',
       'RainRate'],
      dtype='object')
```

```
#new dataframe for models
weather_df = result_df.loc[:, ['Temp', 'Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point']]
weather_df
```

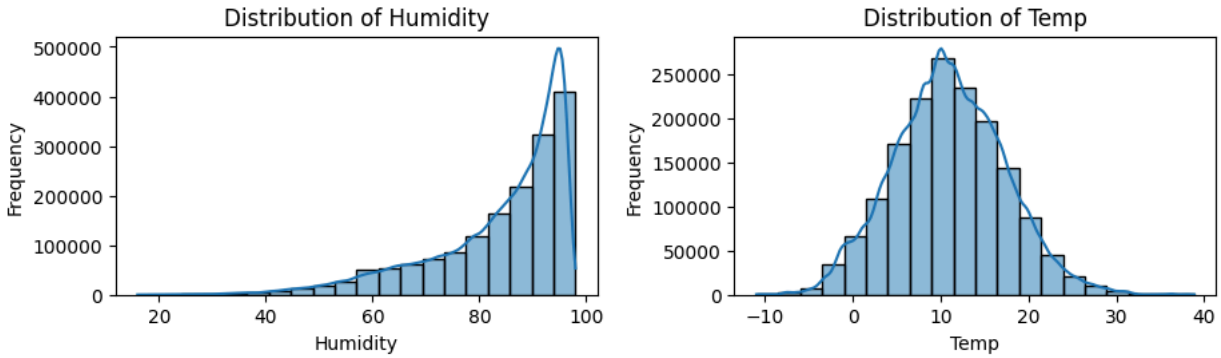
Date	Temp	Humidity	Wind_Spd	Solar	UV	Dew_Point
2021-06-04	14.1	85.0	3.2	0	0.0	11.6
2021-06-04	14.1	86.0	0.0	0	0.0	11.8
2021-06-04	14.1	86.0	3.2	0	0.0	11.8
2021-06-04	14.1	86.0	0.0	0	0.0	11.8
2021-06-04	14.1	86.0	0.0	0	0.0	11.8
...	...	...	...	...	...	...
2021-02-24	11.9	79.0	11.3	0	0.0	8.4
2021-02-24	11.9	79.0	8.0	0	0.0	8.4
2021-02-24	11.9	79.0	6.4	0	0.0	8.4
2021-02-24	11.9	79.0	6.4	0	0.0	8.4
2021-02-24	11.9	79.0	11.3	0	0.0	8.4

1629535 rows × 6 columns

```
weather_df.columns
```

```
Index(['Temp', 'Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point'], dtype='object')
```

```
#Histgram plot for Humidity and Temperature

numerical_variables = [ 'Humidity', 'Temp']

num_bins = 20

plt.figure(figsize=(14, 10))
for i, var in enumerate(numerical_variables, start=1):
    plt.subplot(4, 3, i)
    sns.histplot(result_df[var], bins=num_bins, kde=True)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

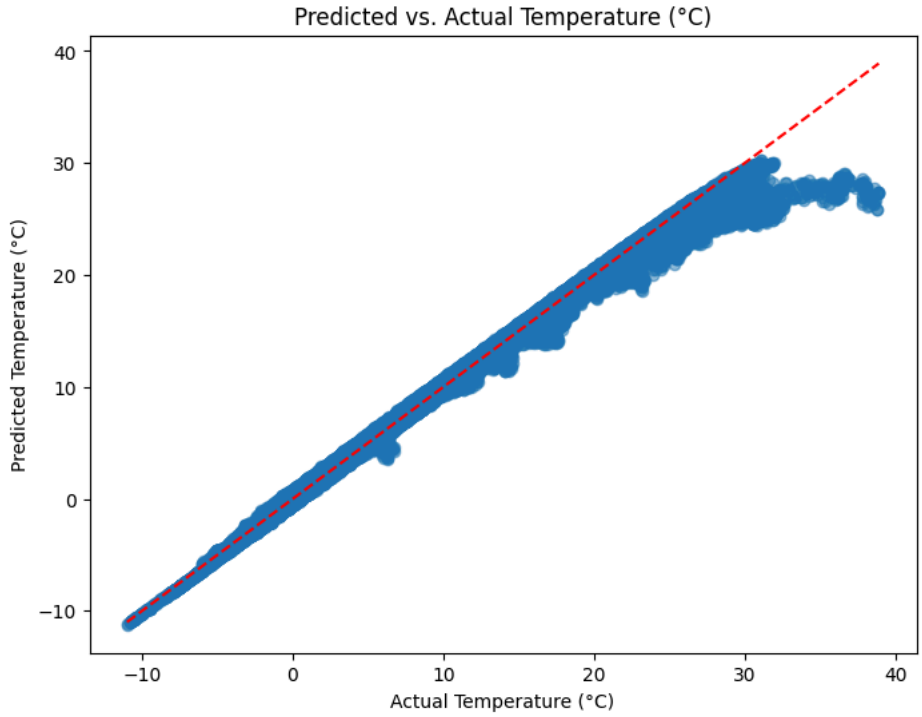



```
#Finding skewness coefficient
skewness = result_df[numerical_variables].skew()
print("Skewness coefficients:")
print(skewness)
```

```
Skewness coefficients:
Humidity   -1.374314
Temp        0.133567
dtype: float64
```

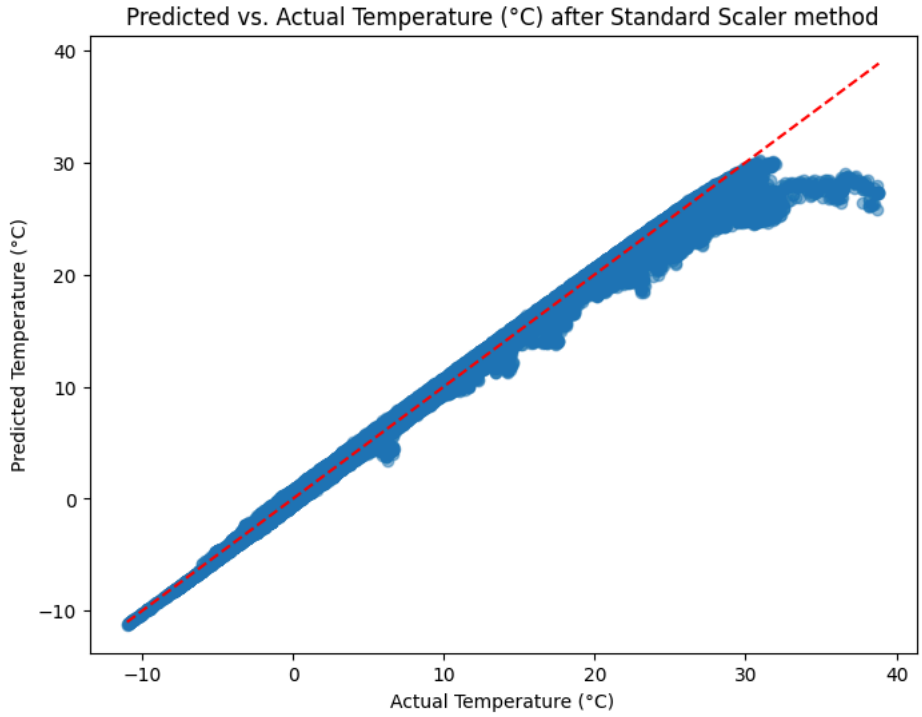
```
#Implementing Linear Regression Model


X = weather_df[['Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point']]
y = weather_df['Temp']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
accuracy_percentage = (1 - mae) * 100
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Accuracy: {accuracy_percentage:.2f}%")


# Visualization
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot(np.linspace(min(y_test), max(y_test), 100), np.linspace(min(y_test), max(y_test), 100), color='red', linestyle='--')
plt.xlabel("Actual Temperature (°C)")
plt.ylabel("Predicted Temperature (°C)")
plt.title("Predicted vs. Actual Temperature (°C)")
plt.show()
```

```
R-squared: 0.9946
Mean Squared Error: 0.22
Root Mean Squared Error: 0.47
Mean Absolute Error: 0.28
Accuracy: 72.03%
```




```
#Linear Regression Model will added Scaler


X = weather_df[['Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point']]
y = weather_df['Temp']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler =  StandardScaler(with_mean=False, with_std=False)
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
accuracy_percentage = (1 - mae) * 100
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Accuracy: {accuracy_percentage:.2f}%")


# Visualization
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot(np.linspace(min(y_test), max(y_test), 100), np.linspace(min(y_test), max(y_test), 100), color='red', linestyle='--')
plt.xlabel("Actual Temperature (°C)")
plt.ylabel("Predicted Temperature (°C)")
plt.title("Predicted vs. Actual Temperature (°C) after Standard Scaler method")
plt.show()
```

```
R-squared: 0.9946
Mean Squared Error: 0.22
Root Mean Squared Error: 0.46
Mean Absolute Error: 0.28
Accuracy: 72.04%
```




```
#Predicting Temperature based on given feature values
#Temp     Humidity  Wind_Spd    Solar   UV  Dew_Point
# 14.1  86.0    3.2 0   0.0 11.8

new_data = [[86.0, 3.2, 0, 0.0, 11.8]]
new_data_scaled = scaler.transform(new_data)
predicted_temp = model.predict(new_data_scaled)
print(f"Predicted temperature for new data: {predicted_temp[0]:.2f} degrees Celsius")
```

```
Predicted temperature for new data: 14.28 degrees Celsius
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler wa
  warnings.warn(
```

```
# Copying specific fields to new dataframe
result_df.reset_index(inplace=True)
prophet_df = result_df[['Time', 'Date', 'Temp', 'Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point']].copy()
sarima_df = result_df[['Time', 'Date', 'Temp', 'Humidity', 'Wind_Spd', 'Solar', 'UV', 'Dew_Point']].copy()
```

```
print(prophet_df.dtypes)
```

```
Time                 object
Date         datetime64[ns]
Temp                float64
Humidity            float64
Wind_Spd            float64
Solar                 int64
UV                  float64
Dew_Point           float64
dtype: object
```

```
# Group the data by 'Date' and aggregate the numeric columns
prophet_df_daily = prophet_df.groupby('Date').agg({
    'Temp': 'mean',
    'Humidity': 'mean',
    'Wind_Spd': 'mean',
    'Solar': 'mean',
    'UV': 'mean',
    'Dew_Point': 'mean'
}).reset_index()


sarima_df_daily = sarima_df.groupby('Date').agg({
    'Temp': 'mean',
    'Humidity': 'mean',
    'Wind_Spd': 'mean',
    'Solar': 'mean',
    'UV': 'mean',
    'Dew_Point': 'mean'
}).reset_index()

# Display the aggregated DataFrame
print(prophet_df_daily)

sarima_df_1 = result_df[['Date','Temp']].copy()

sarima_df_daily_df = sarima_df_1.copy()
sarima_df_daily_df = sarima_df_daily_df.set_index('Date')

sarima_df_daily_df = sarima_df_daily_df.resample('M').mean()
sarima_df_daily_df.head()

sarima_min_daily_df = sarima_df_1.copy()
sarima_min_daily_df = sarima_min_daily_df.set_index('Date')
sarima_min_daily_df = sarima_min_daily_df.resample('M').min()
sarima_min_daily_df.head()

sarima_max_daily_df = sarima_df_1.copy()
sarima_max_daily_df = sarima_max_daily_df.set_index('Date')
sarima_max_daily_df = sarima_max_daily_df.resample('M').max()
sarima_max_daily_df.head()
```

```
           Date       Temp   Humidity   Wind_Spd      Solar        UV  \
0    2021-01-01   1.658611  96.688194   3.527639  11.134722  0.000000
1    2021-01-02   2.515000  92.167361   3.011458  25.375694  0.051667
2    2021-01-03   3.221181  93.111111   4.011181  31.129167  0.060486
3    2021-01-04   3.325278  89.504861   7.778681  11.411111  0.000000
4    2021-01-05   2.897778  89.944444   5.317500  12.791667  0.001389
...         ...        ...        ...        ...        ...       ...
1127 2024-02-02   9.478625  88.859028   7.874514  43.214583  0.141181
1128 2024-02-03  11.661194  88.868056   7.388611  35.130556  0.121944
1129 2024-02-04  11.494556  87.488194  10.451181  31.618750  0.105833
1130 2024-02-05   9.919597  83.017361  12.161042  32.116667  0.100972
1131 2024-02-06  11.233896  85.409028  11.751181  28.233333  0.088264

      Dew_Point
0      1.200347
1      1.359861
2      2.203125
3      1.762847
4      1.414861
...         ...
1127   7.715347
1128   9.880389
1129   9.477271
1130   7.159201
1131   8.843917

[1132 rows x 7 columns]
```

Date	Temp
2021-01-31	12.5
2021-02-28	15.9
2021-03-31	23.3
2021-04-30	16.3
2021-05-31	21.9

Next steps: View recommended plots

```
prophet_df_daily.describe()
```

	Date	Temp	Humidity	Wind_Spd	Solar	UV	Dew_Point
count	1132	1132.000000	1132.000000	1132.000000	1132.000000	1132.000000	1132.000000
mean	2022-07-20 12:00:00	10.985663	83.714185	4.630264	101.865363	0.520613	8.084803
min	2021-01-01 00:00:00	-5.192306	46.528472	0.064444	1.584262	0.000000	-6.711597
25%	2021-10-10 18:00:00	7.001951	78.462500	2.460226	30.959028	0.103976	4.491889
50%	2022-07-20 12:00:00	10.860799	84.592014	3.965243	83.052778	0.400313	8.355056
75%	2023-04-29 06:00:00	15.539705	90.241493	6.143559	157.860590	0.852066	12.117535
max	2024-02-06 00:00:00	28.286438	97.106250	16.283472	308.780556	1.840278	17.788833
std	NaN	5.605392	8.235594	2.939746	79.033293	0.467646	4.979208

```
print(prophet_df_daily.dtypes)
```

```
Date         datetime64[ns]
Temp                float64
Humidity            float64
Wind_Spd            float64
Solar               float64
UV                  float64
Dew_Point           float64
dtype: object
```

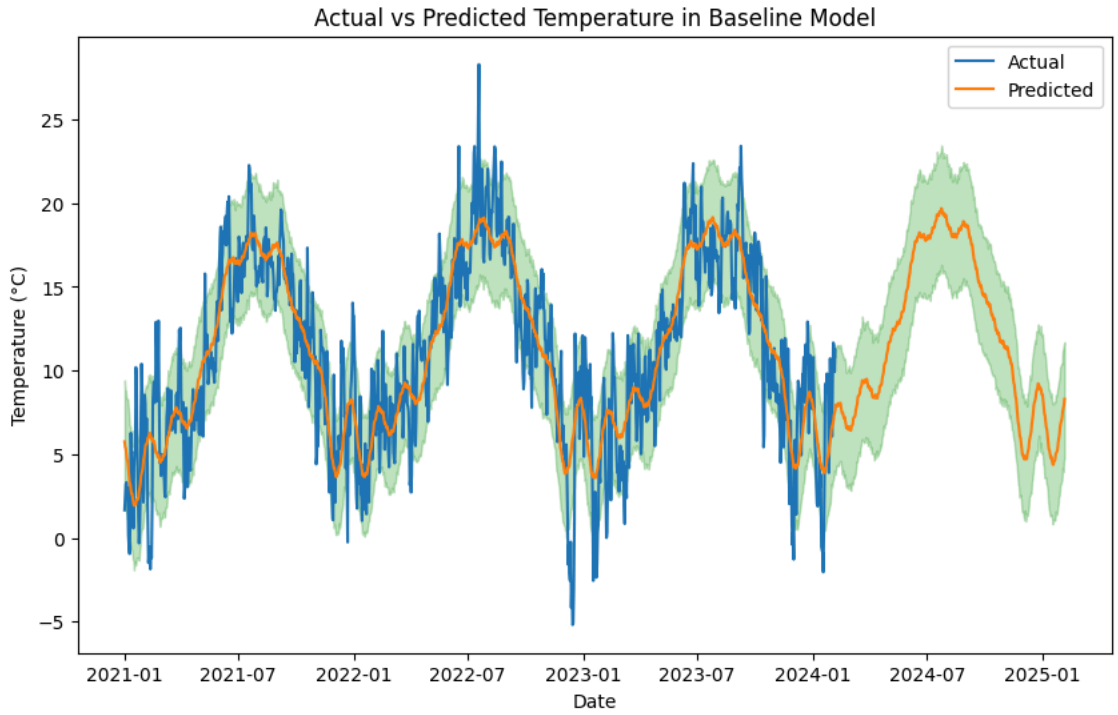
```
pip install prophet
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.2)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.25.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (2.0.3)
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/dist-packages (from prophet) (0.47)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from prophet) (4.66.2)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-packages (from prophet) (6.4.0)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.2.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (4.51
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.4.5
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (3.1.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->holidays>=0.25->prophet)
```

```
#Implementing Prophet Model

prophet_df_daily.rename(columns={'Date': 'ds', 'Temp': 'y'}, inplace=True)


model = Prophet()


model.fit(prophet_df_daily)


future = model.make_future_dataframe(periods=365, freq='D')


forecast = model.predict(future)


print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())


forecast_truncated = forecast.iloc[:len(prophet_df_daily)]

# Calculate R-squared
r2 = r2_score(prophet_df_daily['y'], forecast_truncated['yhat'])

# Calculate Mean Squared Error
mse = mean_squared_error(prophet_df_daily['y'], forecast_truncated['yhat'])

# Calculate Root Mean Squared Error
rmse = mean_squared_error(prophet_df_daily['y'], forecast_truncated['yhat'], squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(prophet_df_daily['y'], forecast_truncated['yhat'])

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpf75oj7yr/iklvm0d0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpf75oj7yr/yc4a03e3.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=10537
20:44:24 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:44:24 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
            ds      yhat  yhat_lower  yhat_upper
1492 2025-02-01  7.163163    3.476860   10.375697
1493 2025-02-02  7.565159    4.164302   11.144477
1494 2025-02-03  7.754352    3.943625   11.346859
1495 2025-02-04  7.919904    4.603229   11.568426
1496 2025-02-05  8.296778    4.789319   11.687229
R-squared: 0.7582
Mean Squared Error: 7.59
Root Mean Squared Error: 2.76
Mean Absolute Error: 2.17
```

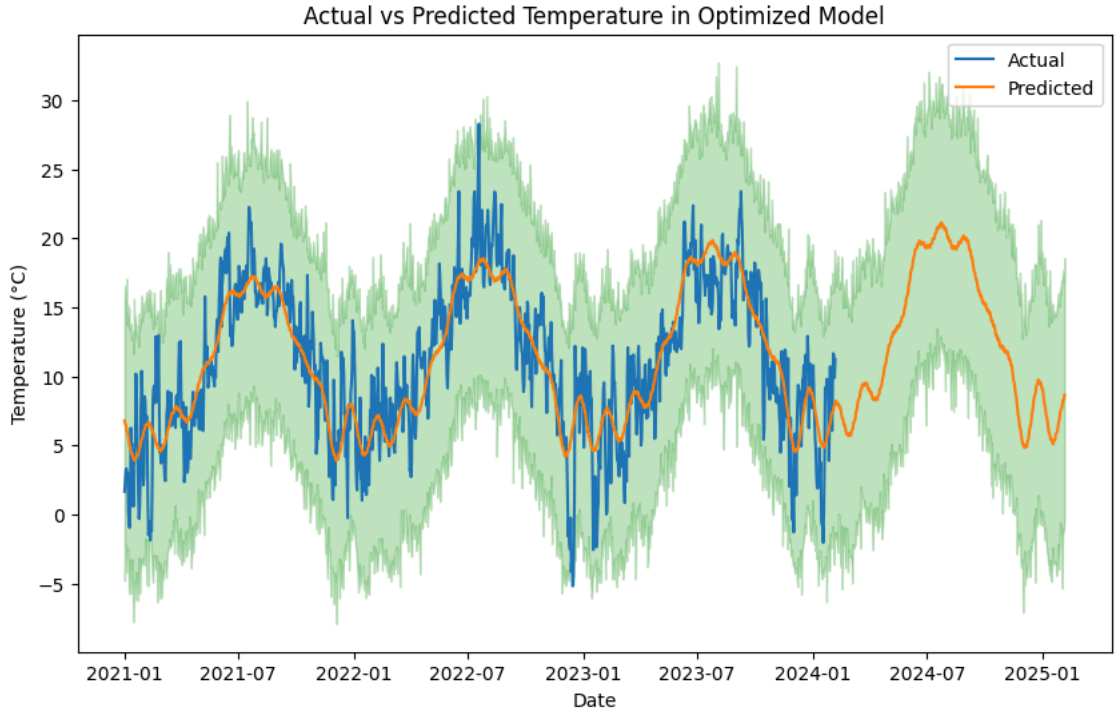
```
plt.figure(figsize=(10, 6))

# Plot actual temperature
sns.lineplot(data=prophet_df_daily, x='ds', y='y', label='Actual', color='#1f77b4')

# Plot predicted temperature
sns.lineplot(data=forecast, x='ds', y='yhat', label='Predicted', color='#ff7f0e')

# Shade the confidence interval for predicted values
plt.fill_between(forecast['ds'], forecast['yhat_lower'], forecast['yhat_upper'], color='#2ca02c', alpha=0.3)


plt.title('Actual vs Predicted Temperature in Baseline Model')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.show()
```



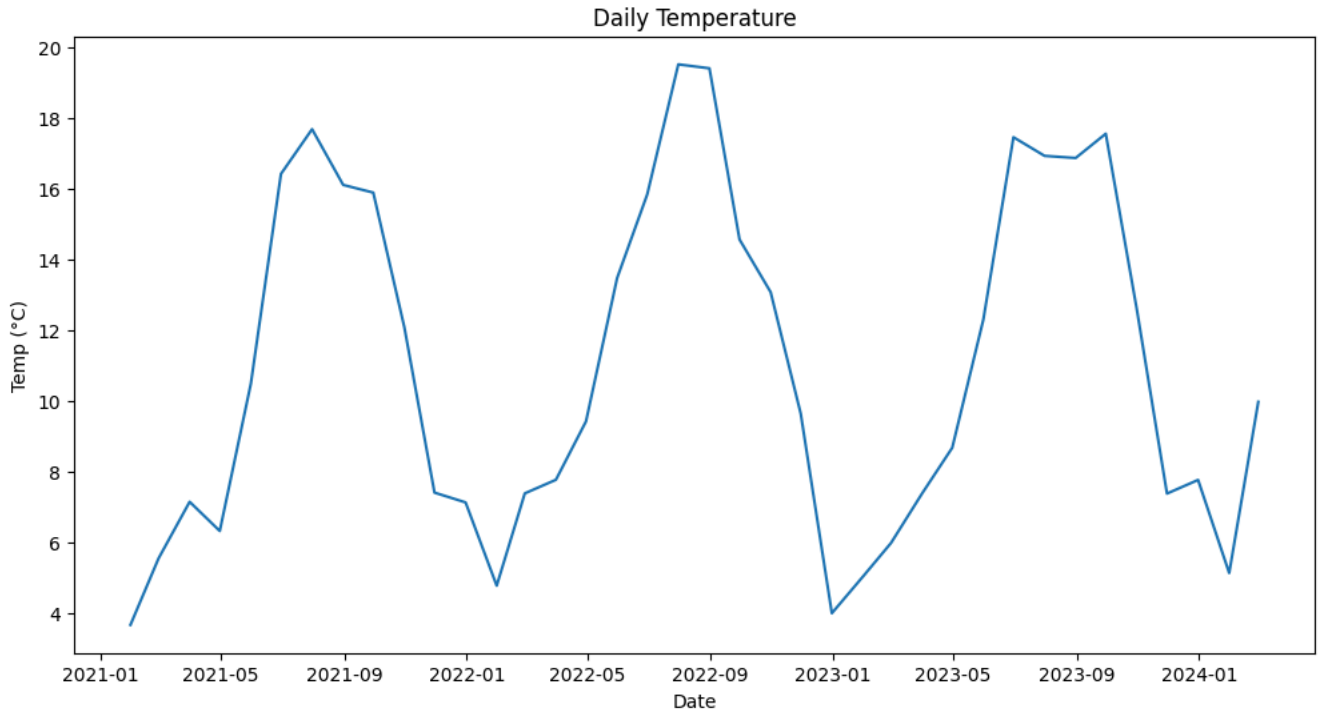

```
# Rename columns to match Prophet's requirements
prophet_df_daily.rename(columns={'Date': 'ds', 'Temp': 'y'}, inplace=True)

# Initialize Prophet model with seasonality adjustments and additional hyperparameters
model = Prophet(
    growth='linear',  # 'linear' or 'logistic' trend
    #yearly_seasonality=True,
    #weekly_seasonality=True,
    daily_seasonality=True,
    seasonality_mode='multiplicative',  # 'additive' or 'multiplicative' seasonality
    changepoint_prior_scale=0.001,  # Controls the flexibility of the trend
    interval_width=1 # Width of the uncertainty intervals (e.g., prediction intervals)
)

# Fit the model
model.fit(prophet_df_daily)

# Generate future dates for forecasting
future = model.make_future_dataframe(periods=365, freq='D')

# Make predictions
forecast = model.predict(future)

# Print the last few rows of the forecast
#print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())


forecast_truncated = forecast.iloc[:len(prophet_df_daily)]

# Calculate R-squared
r2 = r2_score(prophet_df_daily['y'], forecast_truncated['yhat'])

# Calculate Mean Squared Error
mse = mean_squared_error(prophet_df_daily['y'], forecast_truncated['yhat'])

# Calculate Root Mean Squared Error
rmse = mean_squared_error(prophet_df_daily['y'], forecast_truncated['yhat'], squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(prophet_df_daily['y'], forecast_truncated['yhat'])

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")

#R-squared: 0.7582
#Mean Squared Error: 7.59
#Root Mean Squared Error: 2.76
#Mean Absolute Error: 2.17
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmpf75oj7yr/4ek71qx4.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpf75oj7yr/chxa8xe3.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=19481
20:45:55 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:45:55 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
R-squared: 0.7438
Mean Squared Error: 8.04
Root Mean Squared Error: 2.84
Mean Absolute Error: 2.22
```

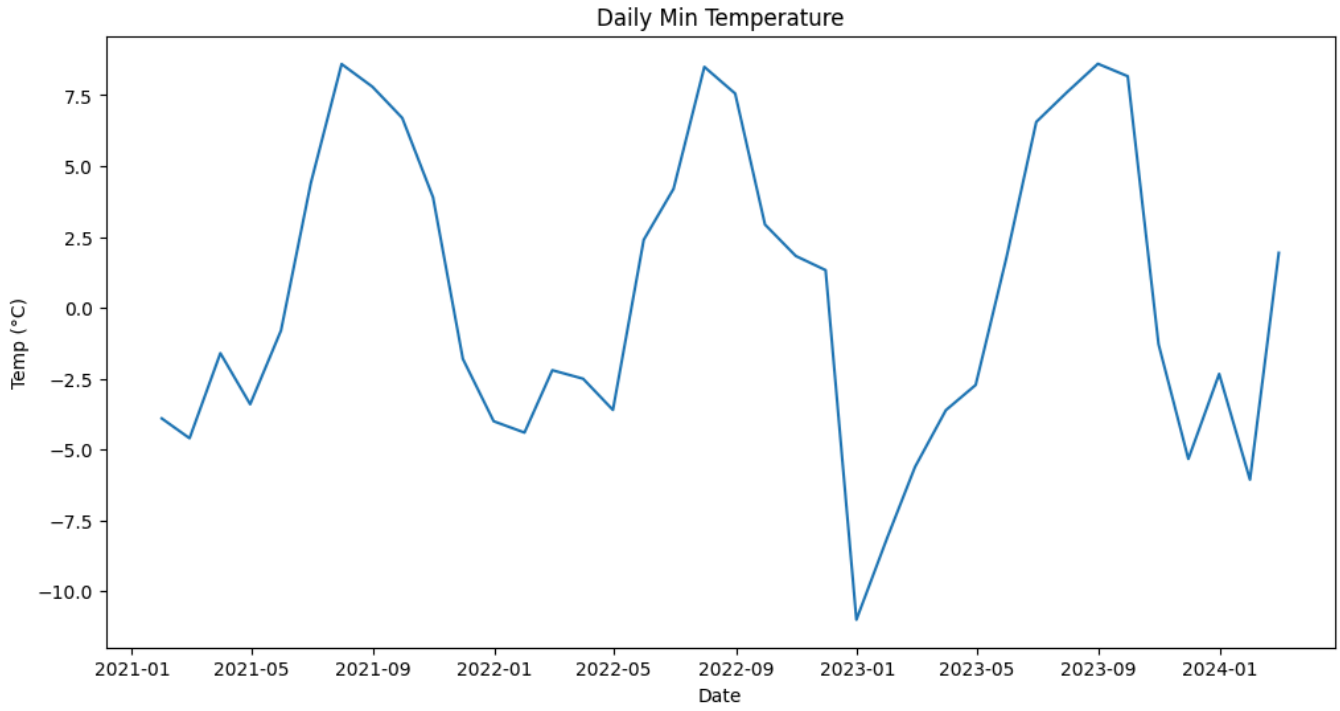
```
plt.figure(figsize=(10, 6))

# Plot actual temperature
sns.lineplot(data=prophet_df_daily, x='ds', y='y', label='Actual', color='#1f77b4')

# Plot predicted temperature
sns.lineplot(data=forecast, x='ds', y='yhat', label='Predicted', color='#ff7f0e')

# Shade the confidence interval for predicted values
plt.fill_between(forecast['ds'], forecast['yhat_lower'], forecast['yhat_upper'], color='#2ca02c', alpha=0.3)


plt.title('Actual vs Predicted Temperature in Optimized Model')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.show()
```

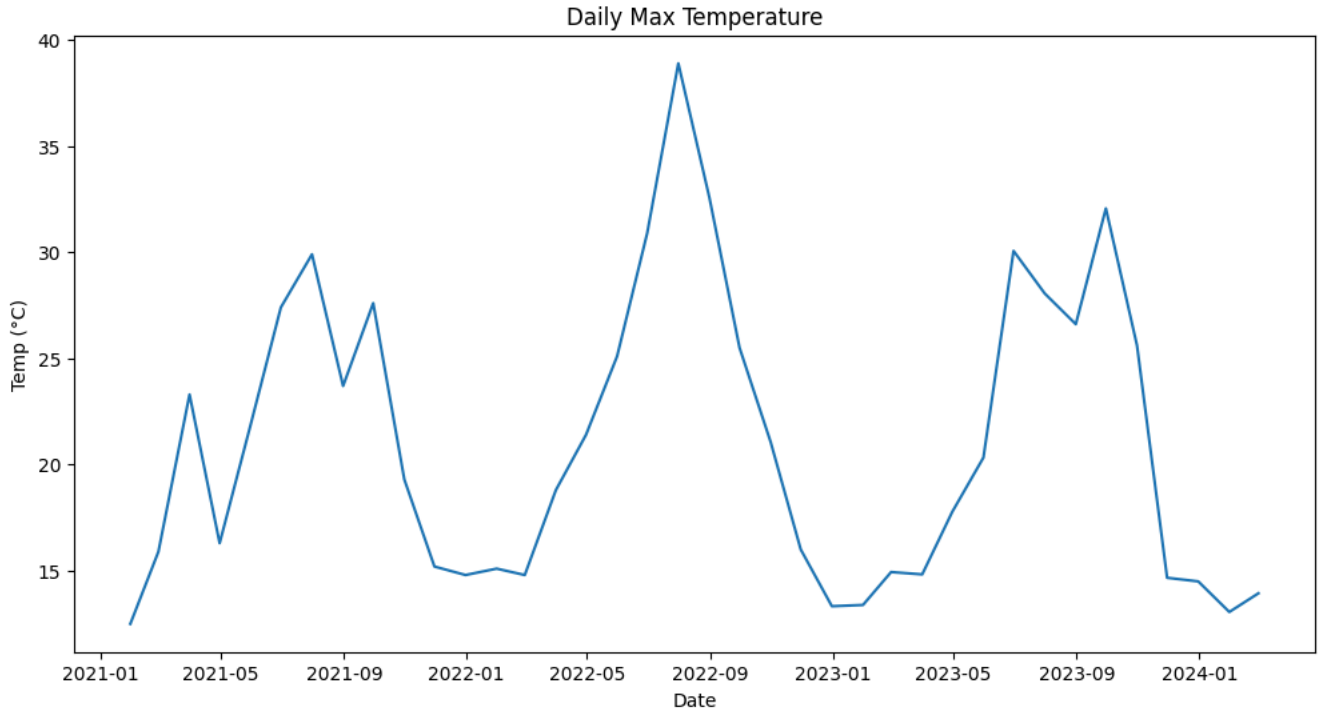



```
plt.figure(figsize=(12, 6))
plt.plot(sarima_df_daily_df)
plt.title("Daily Temperature")
plt.xlabel("Date")
plt.ylabel("Temp (°C)")
plt.show()
```




```
plt.figure(figsize=(12, 6))
plt.plot(sarima_min_daily_df)
plt.title("Daily Min Temperature")
plt.xlabel("Date")
plt.ylabel("Temp (°C)")
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(sarima_max_daily_df)
plt.title("Daily Max Temperature")
plt.xlabel("Date")
plt.ylabel("Temp (°C)")
plt.show()
```



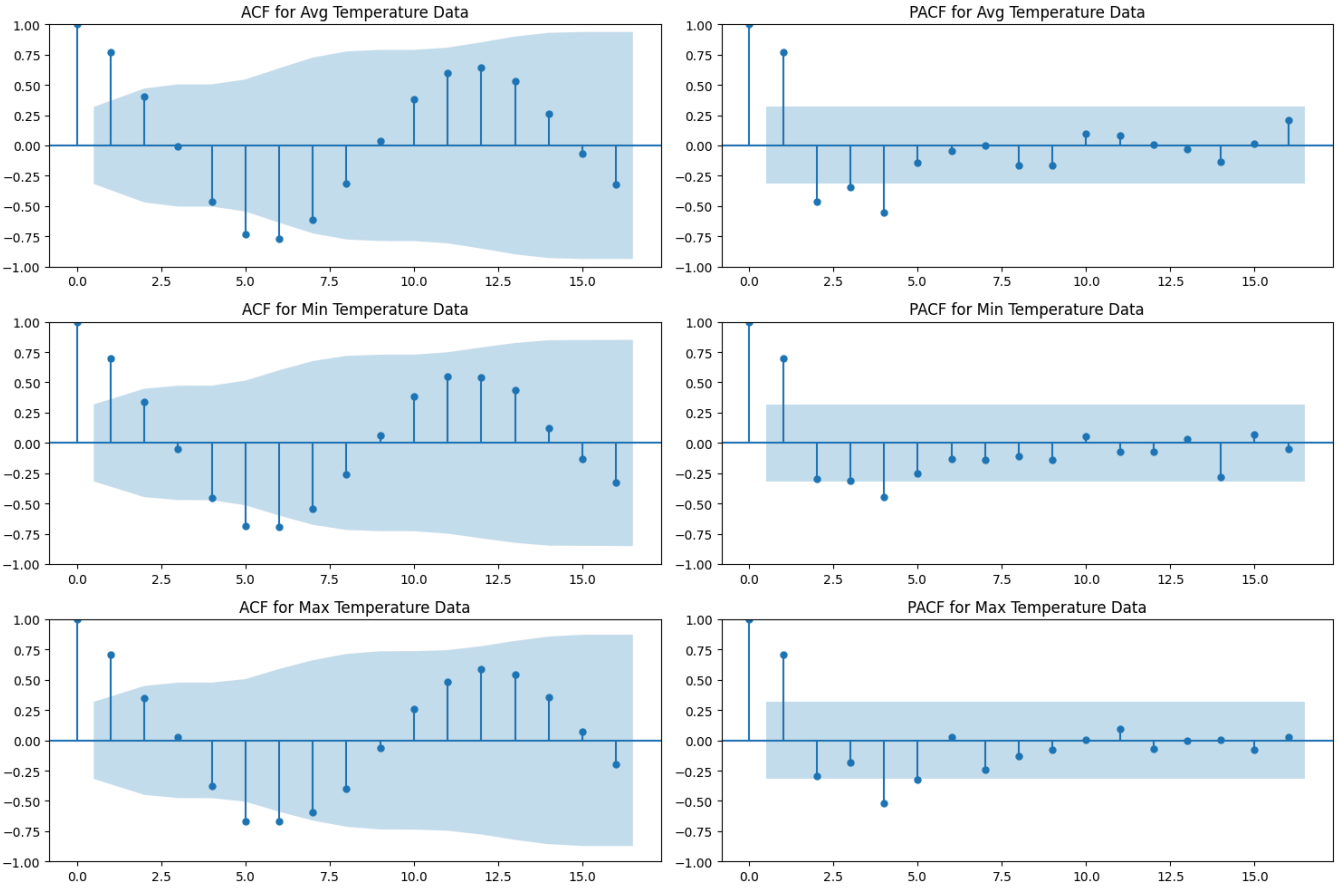



```
def check_stationarity(timeseries):
    # Perform the Dickey-Fuller test
    result = adfuller(timeseries, autolag='AIC')
    p_value = result[1]
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {p_value}')
    print('Stationary' if p_value < 0.05 else 'Non-Stationary')
```

```
check_stationarity(sarima_df_daily_df)
check_stationarity(sarima_min_daily_df)
check_stationarity(sarima_max_daily_df)
```

```
ADF Statistic: -2.888116419555728
p-value: 0.04674745053423251
Stationary
ADF Statistic: -5.454696202327929
p-value: 2.5973940149148112e-06
Stationary
ADF Statistic: -5.963479164475894
p-value: 2.0132300685362882e-07
Stationary
```

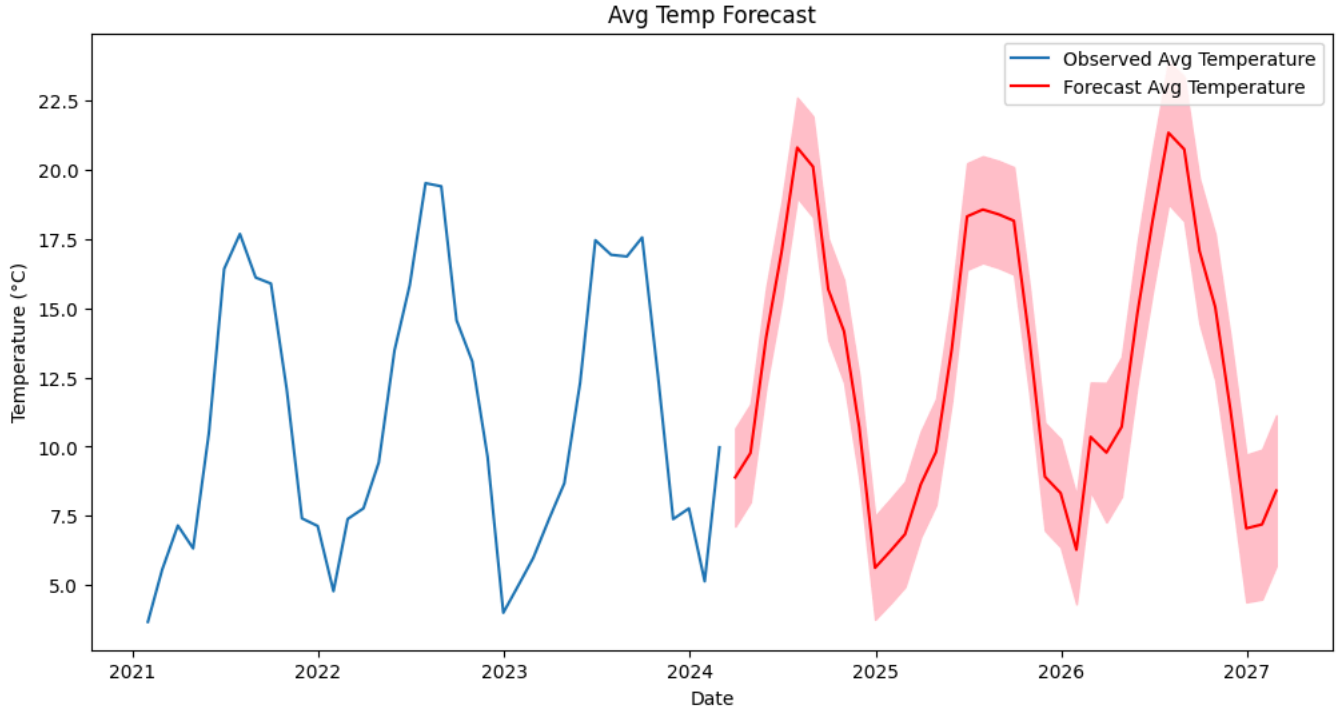
```
plt.figure(figsize=(15, 10))

# Plot ACF and PACF for daily data
plt.subplot(3, 2, 1)
plot_acf(sarima_df_daily_df, ax=plt.gca())
plt.title('ACF for Avg Temperature Data')
plt.subplot(3, 2, 2)
plot_pacf(sarima_df_daily_df, ax=plt.gca())
plt.title('PACF for Avg Temperature Data')

# Plot ACF and PACF for min daily data
plt.subplot(3, 2, 3)
plot_acf(sarima_min_daily_df, ax=plt.gca())
plt.title('ACF for Min Temperature Data')
plt.subplot(3, 2, 4)
plot_pacf(sarima_min_daily_df, ax=plt.gca())
plt.title('PACF for Min Temperature Data')

# Plot ACF and PACF for max daily data
plt.subplot(3, 2, 5)
plot_acf(sarima_max_daily_df, ax=plt.gca())
plt.title('ACF for Max Temperature Data')
plt.subplot(3, 2, 6)
plot_pacf(sarima_max_daily_df, ax=plt.gca())
plt.title('PACF for Max Temperature Data')

plt.tight_layout()
plt.show()
```

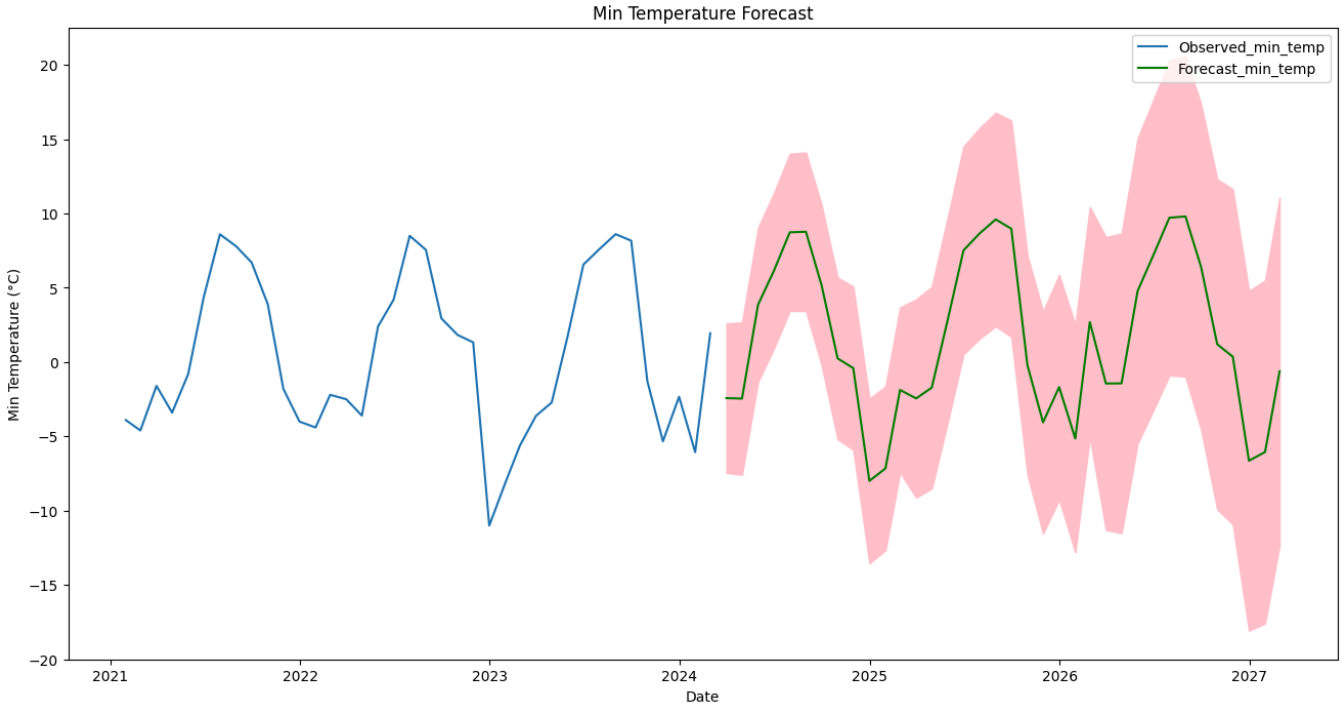

ACF for Avg Temperature Data
PACF for Avg Temperature Data
ACF for Min Temperature Data
PACF for Min Temperature Data
ACF for Max Temperature Data
PACF for Max Temperature Data

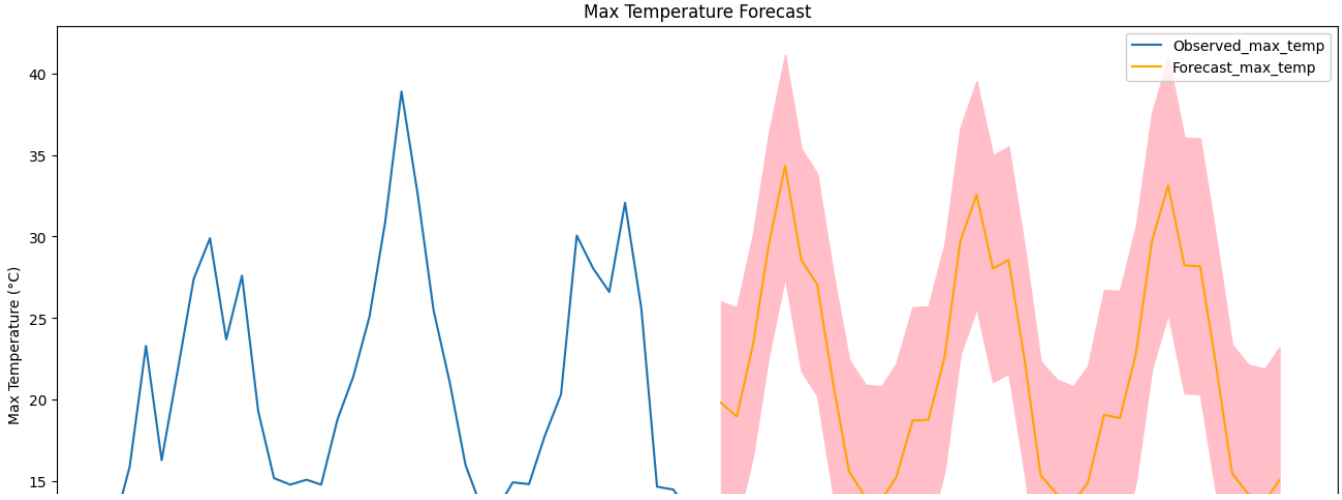
```
p, d, q = 1, 1, 1
P, D, Q, s = 1, 1, 1, 12  # monthly seasonality

# Fit the SARIMA model
model = SARIMAX(sarima_df_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results = model.fit()
model_min = SARIMAX(sarima_min_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results_min = model_min.fit()
model_max = SARIMAX(sarima_max_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results_max = model_max.fit()

# Forecast future values
forecast_periods = 36  # Forecast the next 36 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Plot the forecast
plt.figure(figsize=(12, 6))
plt.plot(sarima_df_daily_df, label='Observed Avg Temperature')
plt.plot(forecast_mean, label='Forecast Avg Temperature', color='red')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.title("Avg Temp Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()

# Forecast min temp future values
forecast_periods_min = 36  # Forecast the next 36 months
forecast_min = results_min.get_forecast(steps=forecast_periods_min)
forecast_mean_min = forecast_min.predicted_mean
forecast_ci_min = forecast_min.conf_int()

# Plot the forecast
plt.figure(figsize=(16, 8))
plt.plot(sarima_min_daily_df, label='Observed_min_temp')
plt.plot(forecast_mean_min, label='Forecast_min_temp', color='green')
plt.fill_between(forecast_ci_min.index, forecast_ci_min.iloc[:, 0], forecast_ci_min.iloc[:, 1], color='pink')
plt.title("Min Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Min Temperature (°C)")
plt.legend()
plt.show()

# Forecast max temp future values
forecast_periods_max = 36  # Forecast the next 36 months
forecast_max = results_max.get_forecast(steps=forecast_periods_max)
forecast_mean_max = forecast_max.predicted_mean
forecast_ci_max = forecast_max.conf_int()

# Plot the forecast
plt.figure(figsize=(16, 8))
plt.plot(sarima_max_daily_df, label='Observed_max_temp')
plt.plot(forecast_mean_max, label='Forecast_max_temp', color='Orange')
plt.fill_between(forecast_ci_max.index, forecast_ci_max.iloc[:, 0], forecast_ci_max.iloc[:, 1], color='pink')
plt.title("Max Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Max Temperature (°C)")
plt.legend()
plt.show()

# Calculate R-squared
r2 = r2_score(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Calculate Mean Squared Error
mse = mean_squared_error(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Calculate Root Mean Squared Error
rmse = mean_squared_error(sarima_df_daily_df[-forecast_periods:], forecast_mean, squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")

# Calculate R-squared
r2_min = r2_score(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)
```

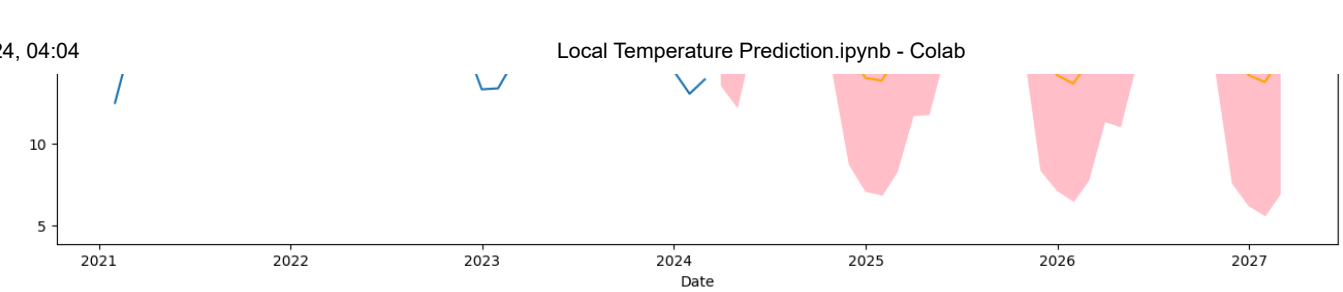
```
# Calculate Mean Squared Error
mse_min = mean_squared_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)

# Calculate Root Mean Squared Error
rmse_min = mean_squared_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min, squared=False)

# Calculate Mean Absolute Error
mae_min = mean_absolute_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)

# Print the calculated metrics
print(f"R-squared Min Temp: {r2_min:.4f}")
print(f"Mean Squared Error Min Temp: {mse_min:.2f}")
print(f"Root Mean Squared Error Min Temp: {rmse_min:.2f}")
print(f"Mean Absolute Error Min Temp: {mae_min:.2f}")

# Calculate R-squared
r2_max = r2_score(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Calculate Mean Squared Error
mse_max = mean_squared_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Calculate Root Mean Squared Error
rmse_max = mean_squared_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max, squared=False)

# Calculate Mean Absolute Error
mae_max = mean_absolute_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Print the calculated metrics
print(f"R-squared Max Temp: {r2_max:.4f}")
print(f"Mean Squared Error Max Temp: {mse_max:.2f}")
print(f"Root Mean Squared Error Max Temp: {rmse_max:.2f}")
print(f"Mean Absolute Error Max Temp: {mae_max:.2f}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
```










```
R-squared: 0.7269
Mean Squared Error: 5.97
Root Mean Squared Error: 2.44
Mean Absolute Error: 2.04
R-squared Min Temp: 0.5904
Mean Squared Error Min Temp: 11.39
Root Mean Squared Error Min Temp: 3.38
Mean Absolute Error Min Temp: 2.56
R-squared Max Temp: 0.8515
Mean Squared Error Max Temp: 6.86
Root Mean Squared Error Max Temp: 2.62
Mean Absolute Error Max Temp: 2.02
```

```
p, d, q = 1, 1, 5
P, D, Q, s = 1, 1, 5, 12  # monthly seasonality

# Fit the SARIMA model
model = SARIMAX(sarima_df_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameter
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to
  warnings.warn("Maximum Likelihood optimization failed to "
```

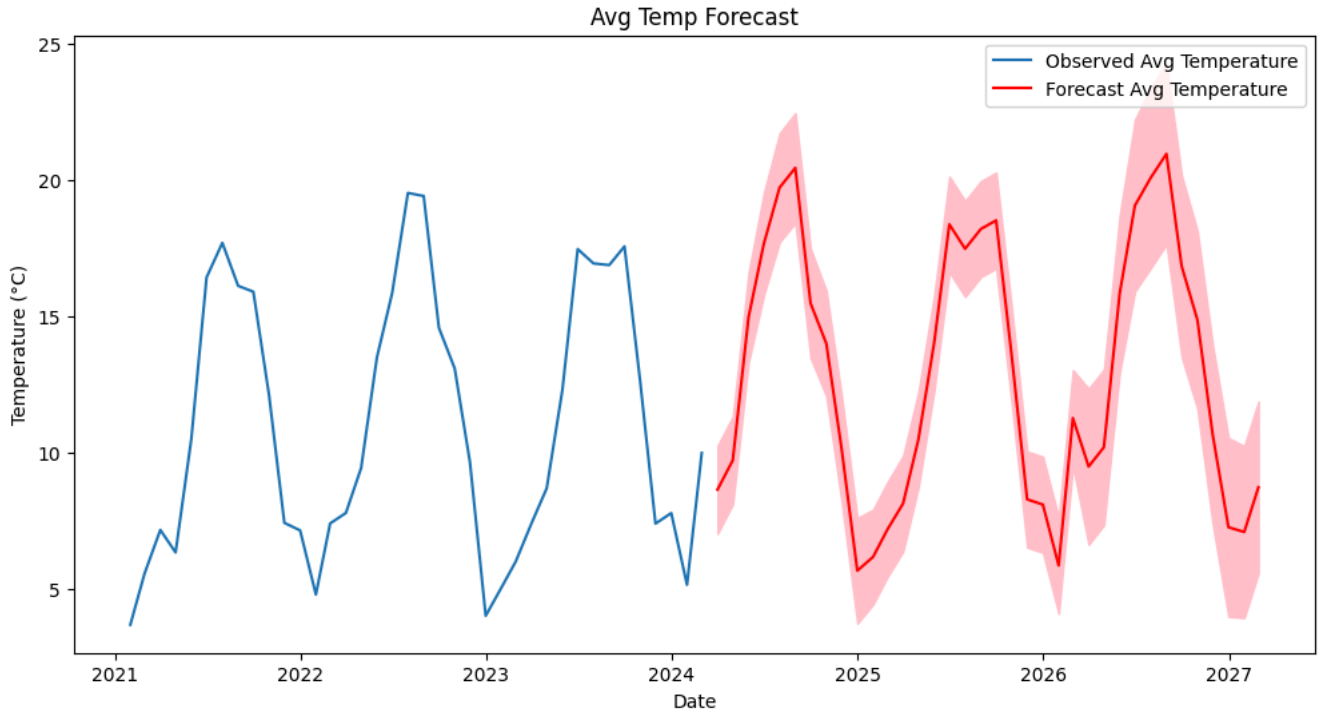
```
p, d, q = 1, 1, 5
P, D, Q, s = 1, 1, 5, 12  # monthly seasonality

# Fit the SARIMA model
model_min = SARIMAX(sarima_min_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results_min = model_min.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameter
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
p, d, q = 1, 1, 5
P, D, Q, s = 1, 1, 5, 12  # monthly seasonality

# Fit the SARIMA model
model_max = SARIMAX(sarima_max_daily_df, order=(p, d, q), seasonal_order=(P, D, Q, s))
results_max = model_max.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to
  warnings.warn("Maximum Likelihood optimization failed to "
```

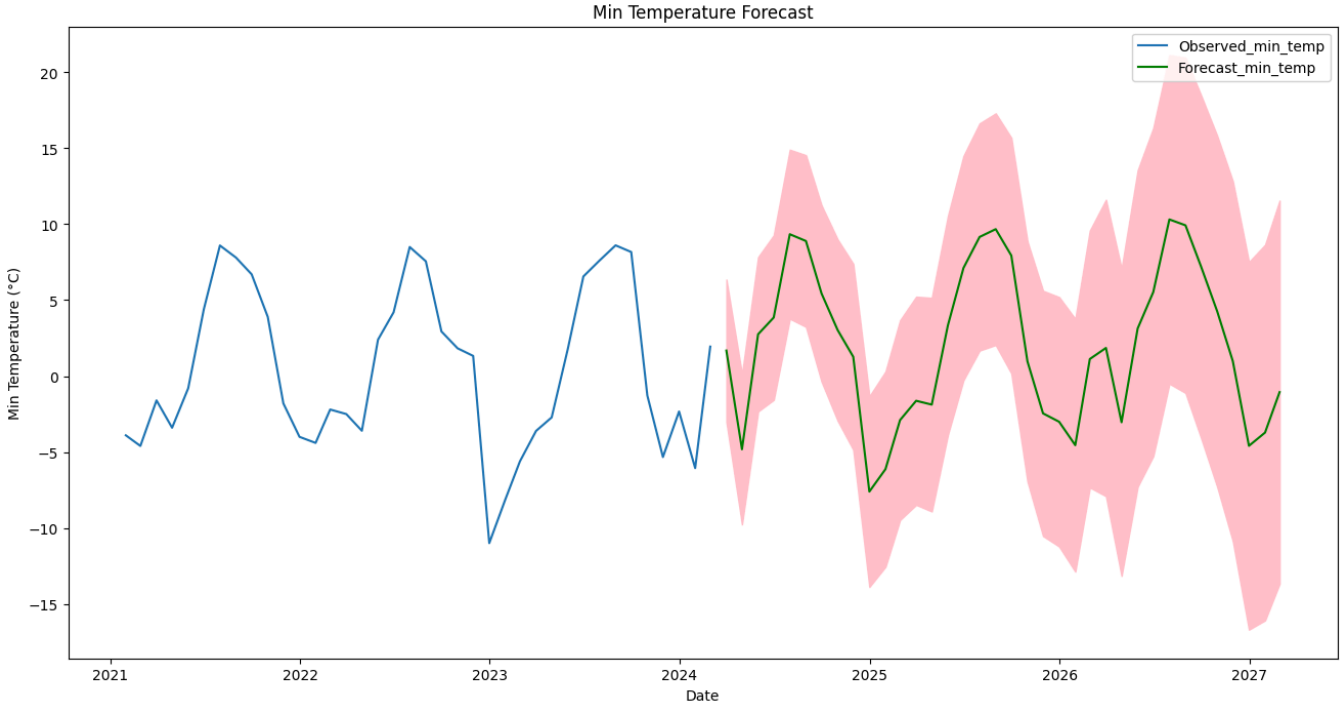
```
# Forecast future values
forecast_periods = 36  # Forecast the next 36 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Plot the forecast
plt.figure(figsize=(12, 6))
plt.plot(sarima_df_daily_df, label='Observed Avg Temperature')
plt.plot(forecast_mean, label='Forecast Avg Temperature', color='red')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.title("Avg Temp Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()

# Calculate R-squared
r2 = r2_score(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Calculate Mean Squared Error
mse = mean_squared_error(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Calculate Root Mean Squared Error
rmse = mean_squared_error(sarima_df_daily_df[-forecast_periods:], forecast_mean, squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(sarima_df_daily_df[-forecast_periods:], forecast_mean)

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```




```
R-squared: 0.7217
Mean Squared Error: 6.08
Root Mean Squared Error: 2.47
Mean Absolute Error: 2.07
```

```
# Forecast min temp future values
forecast_periods_min = 36  # Forecast the next 36 months
forecast_min = results_min.get_forecast(steps=forecast_periods_min)
forecast_mean_min = forecast_min.predicted_mean
forecast_ci_min = forecast_min.conf_int()

# Plot the forecast
plt.figure(figsize=(16, 8))
plt.plot(sarima_min_daily_df, label='Observed_min_temp')
plt.plot(forecast_mean_min, label='Forecast_min_temp', color='green')
```

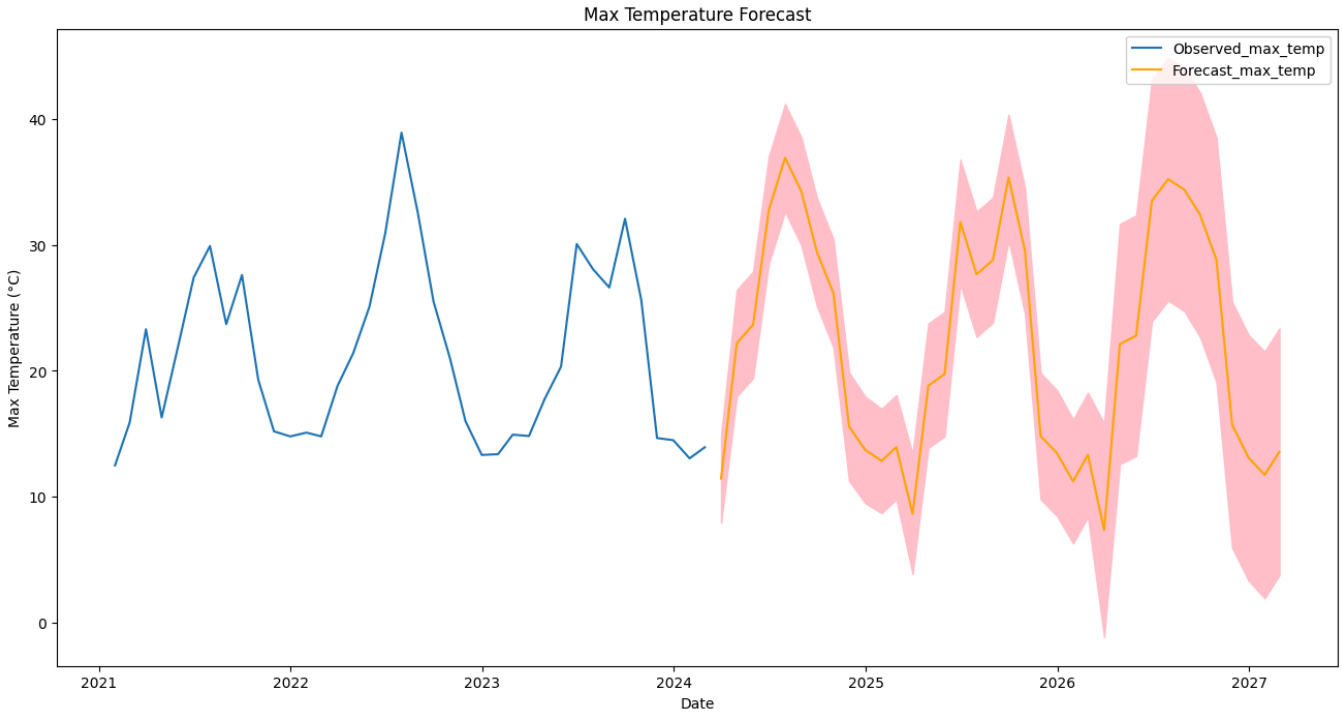
```
plt.plot(forecast_mean_min, label='Forecast_min_temp', color='green')
plt.fill_between(forecast_ci_min.index, forecast_ci_min.iloc[:, 0], forecast_ci_min.iloc[:, 1], color='pink')
plt.title("Min Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Min Temperature (°C)")
plt.legend()
plt.show()

# Calculate R-squared
r2_min = r2_score(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)

# Calculate Mean Squared Error
mse_min = mean_squared_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)

# Calculate Root Mean Squared Error
rmse_min = mean_squared_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min, squared=False)

# Calculate Mean Absolute Error
mae_min = mean_absolute_error(sarima_min_daily_df[-forecast_periods_min:], forecast_mean_min)

# Print the calculated metrics
print(f"R-squared: {r2_min:.4f}")
print(f"Mean Squared Error: {mse_min:.2f}")
print(f"Root Mean Squared Error: {rmse_min:.2f}")
print(f"Mean Absolute Error: {mae_min:.2f}")
```




```
R-squared: 0.6318
Mean Squared Error: 10.24
Root Mean Squared Error: 3.20
Mean Absolute Error: 2.55
```

```
# Forecast max temp future values
forecast_periods_max = 36  # Forecast the next 36 months
forecast_max = results_max.get_forecast(steps=forecast_periods_max)
forecast_mean_max = forecast_max.predicted_mean
forecast_ci_max = forecast_max.conf_int()

# Plot the forecast
plt.figure(figsize=(16, 8))
plt.plot(sarima_max_daily_df, label='Observed_max_temp')
plt.plot(forecast_mean_max, label='Forecast_max_temp', color='Orange')
plt.fill_between(forecast_ci_max.index, forecast_ci_max.iloc[:, 0], forecast_ci_max.iloc[:, 1], color='pink')
plt.title("Max Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Max Temperature (°C)")
plt.legend()
```

```
plt.show()

# Calculate R-squared
r2_max = r2_score(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Calculate Mean Squared Error
mse_max = mean_squared_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Calculate Root Mean Squared Error
rmse_max = mean_squared_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max, squared=False)

# Calculate Mean Absolute Error
mae_max = mean_absolute_error(sarima_max_daily_df[-forecast_periods_max:], forecast_mean_max)

# Print the calculated metrics
print(f"R-squared: {r2_max:.4f}")
print(f"Mean Squared Error: {mse_max:.2f}")
print(f"Root Mean Squared Error: {rmse_max:.2f}")
print(f"Mean Absolute Error: {mae_max:.2f}")
```



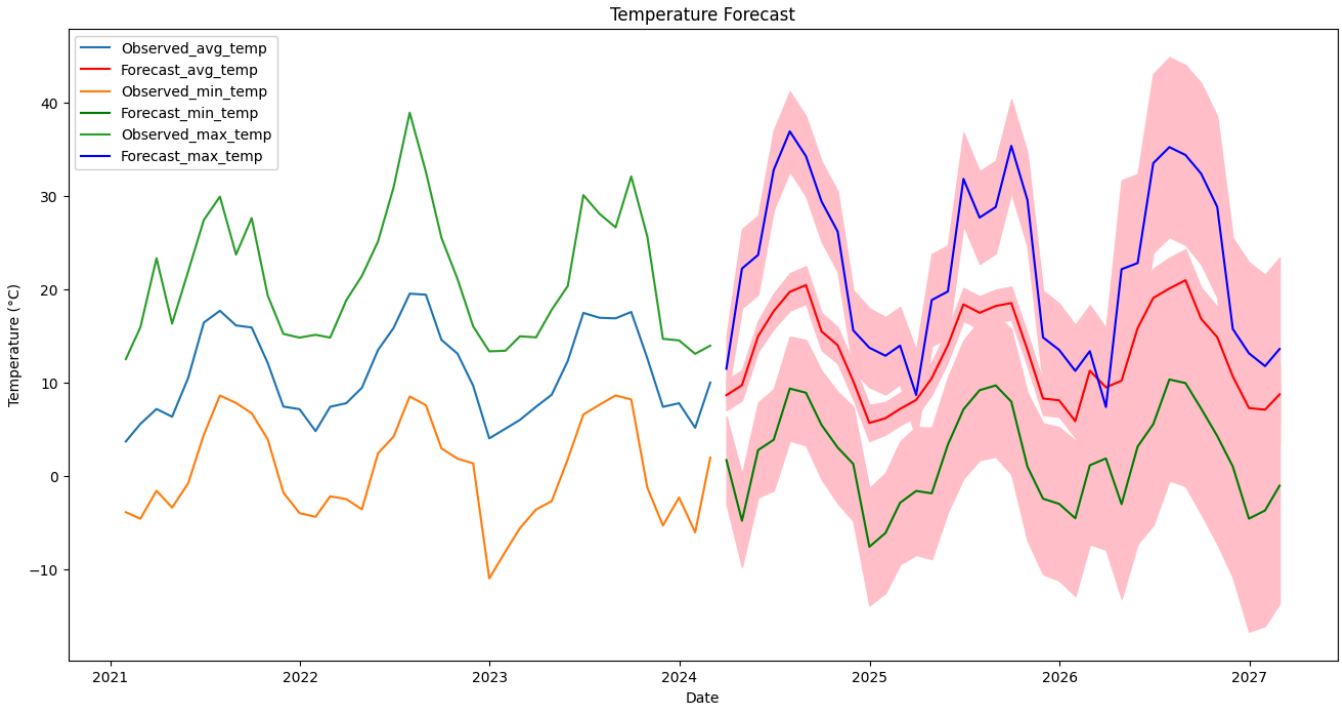

```
R-squared: 0.3389
Mean Squared Error: 30.54
Root Mean Squared Error: 5.53
Mean Absolute Error: 4.25
```

```
pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 10.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.4.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.10)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.25.2)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.0.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.11.4)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (24.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1
Installing collected packages: pmdarima
```

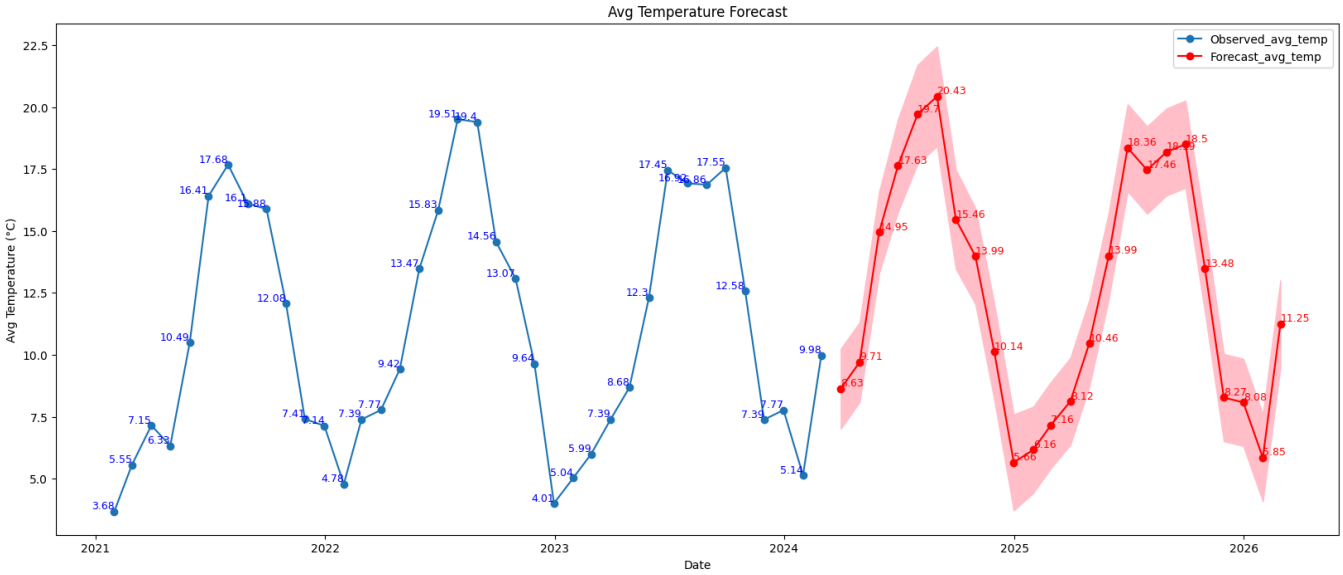
```
Successfully installed pmdarima-2.0.4
```

```
# Forecast avg temp future values
forecast_periods = 36  # Forecast the next 36 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Forecast min temp future values
forecast_periods_min = 36  # Forecast the next 36 months
forecast_min = results_min.get_forecast(steps=forecast_periods_min)
forecast_mean_min = forecast_min.predicted_mean
forecast_ci_min = forecast_min.conf_int()

# Forecast max temp future values
forecast_periods_max = 36  # Forecast the next 36 months
forecast_max = results_max.get_forecast(steps=forecast_periods_max)
forecast_mean_max = forecast_max.predicted_mean
forecast_ci_max = forecast_max.conf_int()

# Plot the forecast
plt.figure(figsize=(16, 8))
plt.plot(sarima_df_daily_df, label='Observed_avg_temp')
plt.plot(forecast_mean, label='Forecast_avg_temp', color='red')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.plot(sarima_min_daily_df, label='Observed_min_temp')
plt.plot(forecast_mean_min, label='Forecast_min_temp', color='green')
plt.fill_between(forecast_ci_min.index, forecast_ci_min.iloc[:, 0], forecast_ci_min.iloc[:, 1], color='pink')
plt.plot(sarima_max_daily_df, label='Observed_max_temp')
plt.plot(forecast_mean_max, label='Forecast_max_temp', color='Blue')
plt.fill_between(forecast_ci_max.index, forecast_ci_max.iloc[:, 0], forecast_ci_max.iloc[:, 1], color='pink')
plt.title("Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()
```

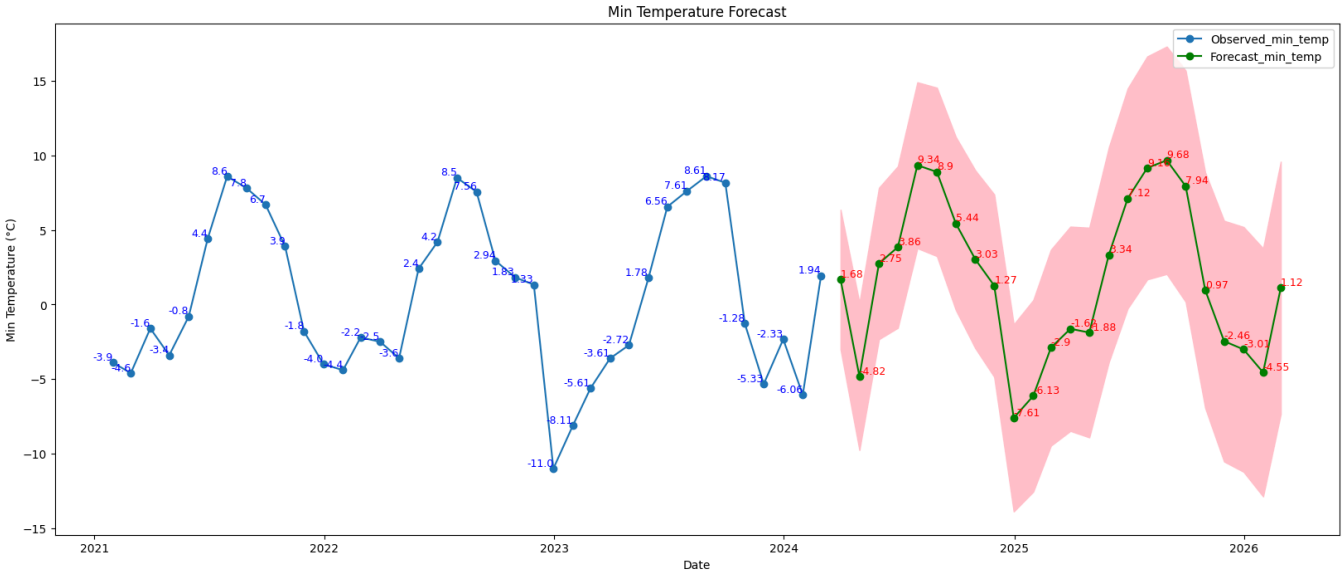



```
# Forecast future values
forecast_periods = 24  # Forecast the next 24 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()
```

```
# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(sarima_df_daily_df.index, sarima_df_daily_df['Temp'], label='Observed_avg_temp', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast_avg_temp', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')

plt.title("Avg Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Avg Temperature (°C)")
plt.legend()


# plot data points with their values
for i, value in enumerate(sarima_df_daily_df['Temp']):
    plt.text(sarima_df_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean):
    plt.text(forecast_mean.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```

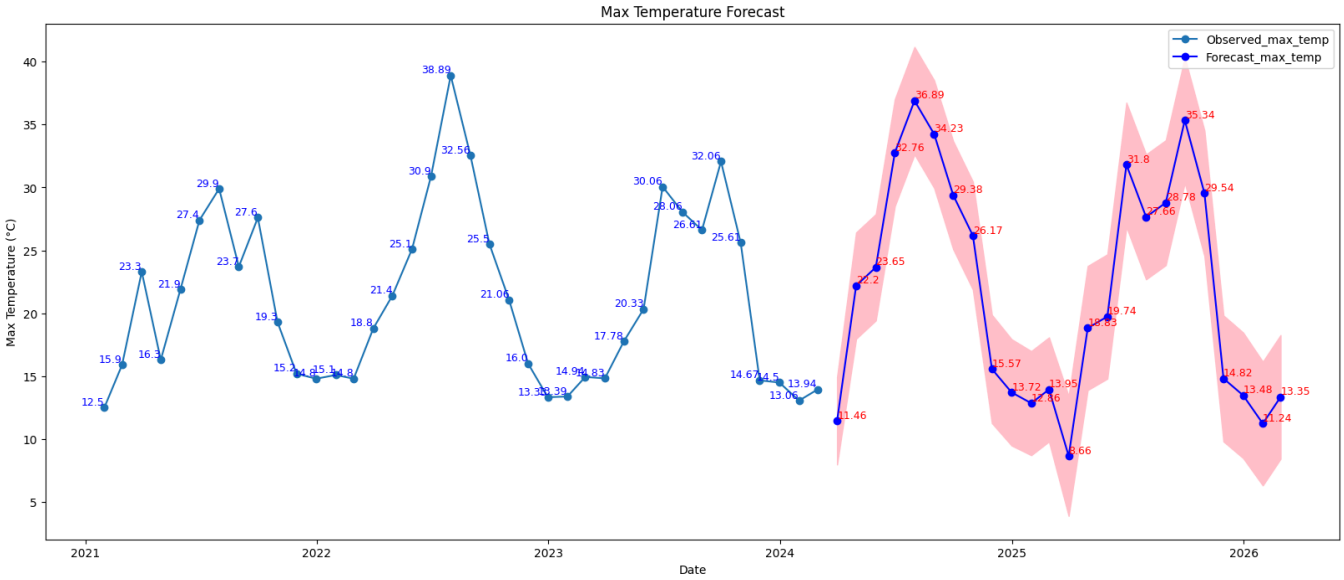


```
# Forecast min temp future values
forecast_periods_min = 24  # Forecast the next 24 months
forecast_min = results_min.get_forecast(steps=forecast_periods_min)
forecast_mean_min = forecast_min.predicted_mean
forecast_ci_min = forecast_min.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(sarima_min_daily_df.index, sarima_min_daily_df['Temp'], label='Observed_min_temp', marker='o')
plt.plot(forecast_mean_min.index, forecast_mean_min, label='Forecast_min_temp', color='Green', marker='o')
plt.fill_between(forecast_ci_min.index, forecast_ci_min.iloc[:, 0], forecast_ci_min.iloc[:, 1], color='pink')

plt.title("Min Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Min Temperature (°C)")
plt.legend()

    # plot data points with their values
for i, value in enumerate(sarima_min_daily_df['Temp']):
    plt.text(sarima_min_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean_min):
    plt.text(forecast_mean_min.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```

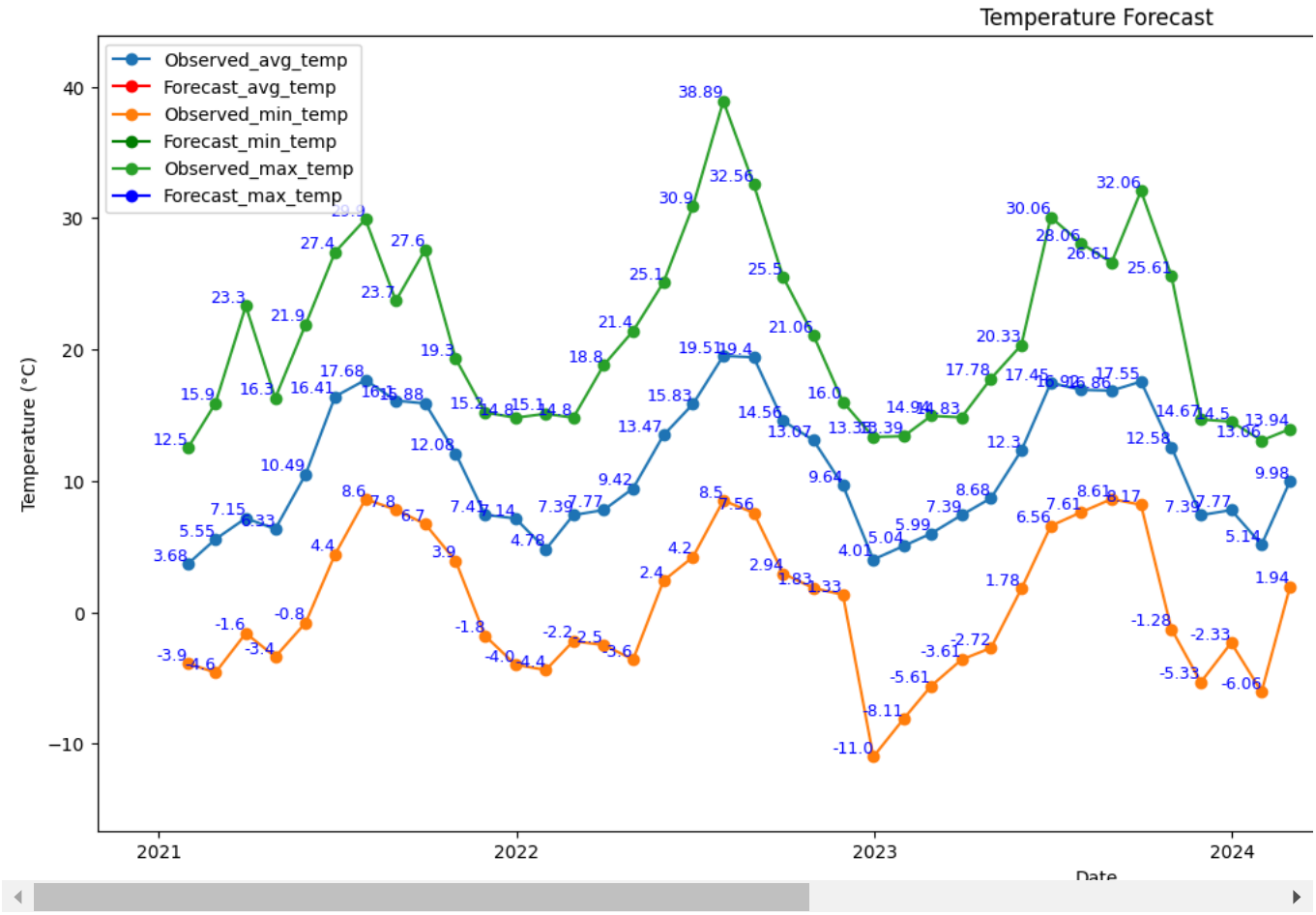



```
# Forecast max temp future values
forecast_periods_max = 24  # Forecast the next 24 months
forecast_max = results_max.get_forecast(steps=forecast_periods_max)
forecast_mean_max = forecast_max.predicted_mean
forecast_ci_max = forecast_max.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))

plt.plot(sarima_max_daily_df.index, sarima_max_daily_df['Temp'], label='Observed_max_temp', marker='o')
plt.plot(forecast_mean_max.index, forecast_mean_max, label='Forecast_max_temp', color='Blue', marker='o')
plt.fill_between(forecast_ci_max.index, forecast_ci_max.iloc[:, 0], forecast_ci_max.iloc[:, 1], color='pink')

plt.title("Max Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Max Temperature (°C)")
plt.legend()
```

```
        # plot data points with their values
for i, value in enumerate(sarima_max_daily_df['Temp']):
    plt.text(sarima_max_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean_max):
    plt.text(forecast_mean_max.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```



```
# Forecast future values
forecast_periods = 24  # Forecast the next 24 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Forecast min temp future values
forecast_periods_min = 24  # Forecast the next 36 months
forecast_min = results_min.get_forecast(steps=forecast_periods_min)
forecast_mean_min = forecast_min.predicted_mean
forecast_ci_min = forecast_min.conf_int()

# Forecast max temp future values
forecast_periods_max = 24  # Forecast the next 36 months
forecast_max = results_max.get_forecast(steps=forecast_periods_max)
forecast_mean_max = forecast_max.predicted_mean
forecast_ci_max = forecast_max.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(sarima_df_daily_df.index, sarima_df_daily_df['Temp'], label='Observed_avg_temp', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast_avg_temp', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')

plt.plot(sarima_min_daily_df.index, sarima_min_daily_df['Temp'], label='Observed_min_temp', marker='o')
plt.plot(forecast_mean_min.index, forecast_mean_min, label='Forecast_min_temp', color='Green', marker='o')
plt.fill_between(forecast_ci_min.index, forecast_ci_min.iloc[:, 0], forecast_ci_min.iloc[:, 1], color='pink')

plt.plot(sarima_max_daily_df.index, sarima_max_daily_df['Temp'], label='Observed_max_temp', marker='o')
plt.plot(forecast_mean_max.index, forecast_mean_max, label='Forecast_max_temp', color='Blue', marker='o')
plt.fill_between(forecast_ci_max.index, forecast_ci_max.iloc[:, 0], forecast_ci_max.iloc[:, 1], color='pink')

plt.title("Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()


# plot data points with their values
for i, value in enumerate(sarima_df_daily_df['Temp']):
    plt.text(sarima_df_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean):
    plt.text(forecast_mean.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

    # plot data points with their values
for i, value in enumerate(sarima_min_daily_df['Temp']):
    plt.text(sarima_min_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean_min):
    plt.text(forecast_mean_min.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')


        # plot data points with their values
for i, value in enumerate(sarima_max_daily_df['Temp']):
    plt.text(sarima_max_daily_df.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# plot data points with their values
for i, value in enumerate(forecast_mean_max):
    plt.text(forecast_mean_max.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```



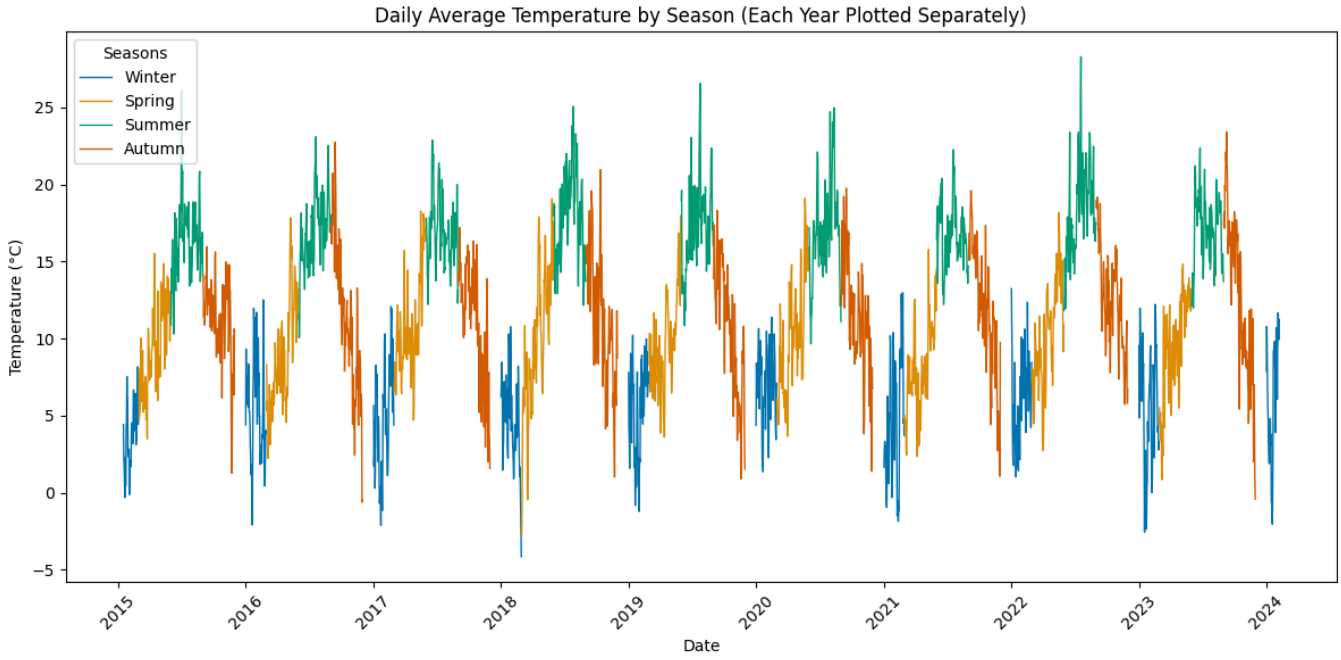

```
#loading full data set from google drive
folder_path = '/content/drive/MyDrive/WeatherData'

file_list = os.listdir(folder_path)
full_dfs = []
for file in file_list:
    if file.endswith('.txt'):
        file_path = os.path.join(folder_path, file)
        full_df = pd.read_csv(file_path, skiprows=3, delimiter='\s+', encoding='utf-8', header=None)
        with open(file_path, 'r', encoding='latin1') as f:
            first_line = f.readline().strip()
        date_value = first_line.split()[0]
        full_df['Date'] = pd.to_datetime(date_value, format='%d/%m/%y')
        full_dfs.append(full_df)

        full_result_df = pd.concat(full_dfs, ignore_index=True)


#assigining column names to the dataframe
with open('/content/drive/MyDrive/Test/column_names.txt', 'r') as file:
    column_names = file.readline().strip().split()

full_result_df.columns = column_names


full_result_df.tail()
```

	Time	Wind_Dir	Wind_Spd	Wind_Gust	Hum_In	Humidity	Temp_In	Temp	Raw_Barom	UV	Solar	Dew_Point	DailyRain	RainRate
4754973	23:55	4.0	1.6	4.8	41.0	88.0	19.7	6.1	1027.0	0.0	0	4.2	0.0	0.0
4754974	23:56	50.0	3.2	6.4	41.0	88.0	19.7	6.1	1027.0	0.0	0	4.2	0.0	0.0
4754975	23:57	4.0	0.0	4.8	41.0	88.0	19.7	6.1	1027.0	0.0	0	4.2	0.0	0.0

```
# Create a new column 'Year' containing only the year values
full_result_df['Year'] = full_result_df['Date'].dt.year

# Display the DataFrame with the new 'Year' column
print(full_result_df.head())
```

```
   Time  Wind_Dir  Wind_Spd  Wind_Gust  Hum_In  Humidity  Temp_In  Temp  \
0  0:00      67.0       0.0        0.0    48.0      88.0     10.2   1.7
1  0:01      66.0       0.0        0.0    48.0      88.0     10.2   1.6
2  0:02      67.0       0.0        0.0    48.0      88.0     10.2   1.6
3  0:03      67.0       0.0        0.0    48.0      88.0     10.2   1.6
4  0:04      66.0       0.0        0.0    48.0      88.0     10.2   1.6

   Raw_Barom   UV  Solar  Dew_Point  DailyRain  RainRate       Date  Year
0     1022.4  0.0      0       -0.1        0.0       0.0 2021-04-23  2021
1     1022.4  0.0      0       -0.2        0.0       0.0 2021-04-23  2021
2     1022.3  0.0      0       -0.2        0.0       0.0 2021-04-23  2021
3     1022.3  0.0      0       -0.2        0.0       0.0 2021-04-23  2021
4     1022.4  0.0      0       -0.2        0.0       0.0 2021-04-23  2021
```

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

full_result_df.set_index('Date', inplace=True)

# Calculate daily average temperature
daily_avg_temp = full_result_df['Temp'].resample('D').mean()

# Convert daily_avg_temp to a DataFrame with 'Date' as a column
daily_avg_temp_df = daily_avg_temp.to_frame()

daily_avg_temp_df.reset_index(inplace=True)

# Define the seasons based on the month
seasons = ['Winter', 'Spring', 'Summer', 'Autumn']
season_bins = [0, 3, 6, 9, 12]  # January-March, April-June, July-September, October-December
daily_avg_temp_df['Season'] = pd.cut(daily_avg_temp_df['Date'].dt.month, bins=season_bins, labels=seasons, right=False)

# Plot daily average temperature by season for each year separately
plt.figure(figsize=(12, 6), dpi=100)
for year in daily_avg_temp_df['Date'].dt.year.unique():
    year_data = daily_avg_temp_df[daily_avg_temp_df['Date'].dt.year == year]
    sns.lineplot(data=year_data, x='Date', y='Temp', hue='Season', palette='colorblind', linewidth=1)  # Adjusting linewidth

# Create custom legend for the four seasons
handles, labels = plt.gca().get_legend_handles_labels()
season_handles = handles[:4]
season_labels = labels[:4]
plt.legend(season_handles, season_labels, title='Seasons', loc='upper left')

plt.title('Daily Average Temperature by Season (Each Year Plotted Separately)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



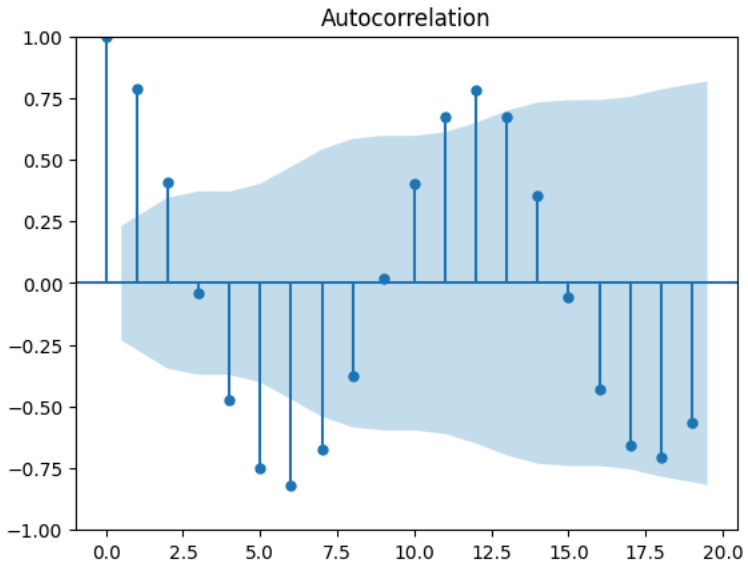
```
full_sarima_df_train = full_result_df[full_result_df['Year'] <= 2020]
full_sarima_df_validate = full_result_df[full_result_df['Year'] >= 2021]

full_sarima_df_train.reset_index(inplace=True)
full_sarima_df_validate.reset_index(inplace=True)

# Select 'Date' and 'Temp' columns and set 'Date' as index
full_sarima_df_train = full_sarima_df_train[['Date', 'Temp']].set_index('Date')
full_sarima_df_test = full_sarima_df_validate[['Date', 'Temp']].set_index('Date')

# Resample and calculate mean for training data
full_sarima_df_train = full_sarima_df_train.resample('M').mean()

# Resample and calculate mean for validation data
full_sarima_df_test = full_sarima_df_test.resample('M').mean()

# Print the head of the resampled training and test data
print(full_sarima_df_train.head())
print(full_sarima_df_test.head())
```

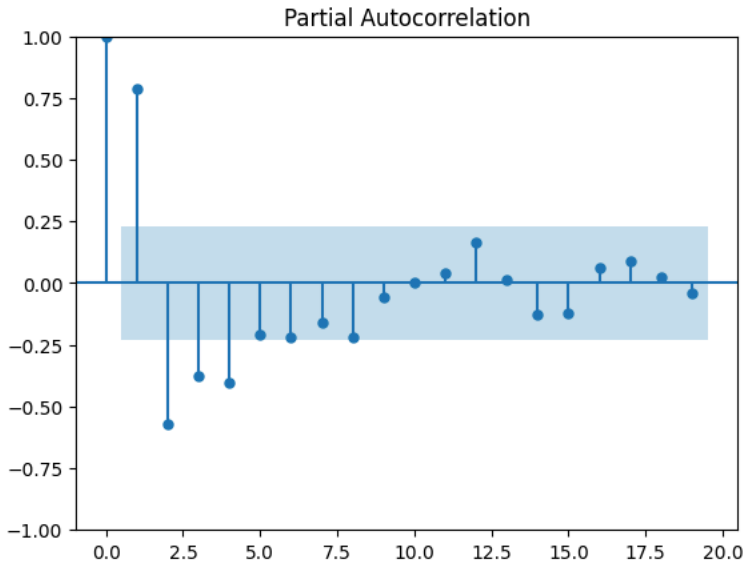
```
                Temp
Date
2015-01-31   2.979431
2015-02-28   4.355620
2015-03-31   6.939719
2015-04-30   9.275339
2015-05-31  11.679865
                Temp
Date
2021-01-31   3.676884
2021-02-28   5.554245
2021-03-31   7.153284
2021-04-30   6.328137
2021-05-31  10.491929
```

```
print(full_sarima_df_train.tail())
print(full_sarima_df_test.tail())
```

```
                Temp
Date
2020-08-31  18.473273
2020-09-30  14.573176
2020-10-31  10.882119
2020-11-30   8.683351
2020-12-31   5.539539
                Temp
Date
2023-10-31  12.577212
2023-11-30   7.385004
2023-12-31   7.770378
2024-01-31   5.141734
2024-02-29   9.975809
```

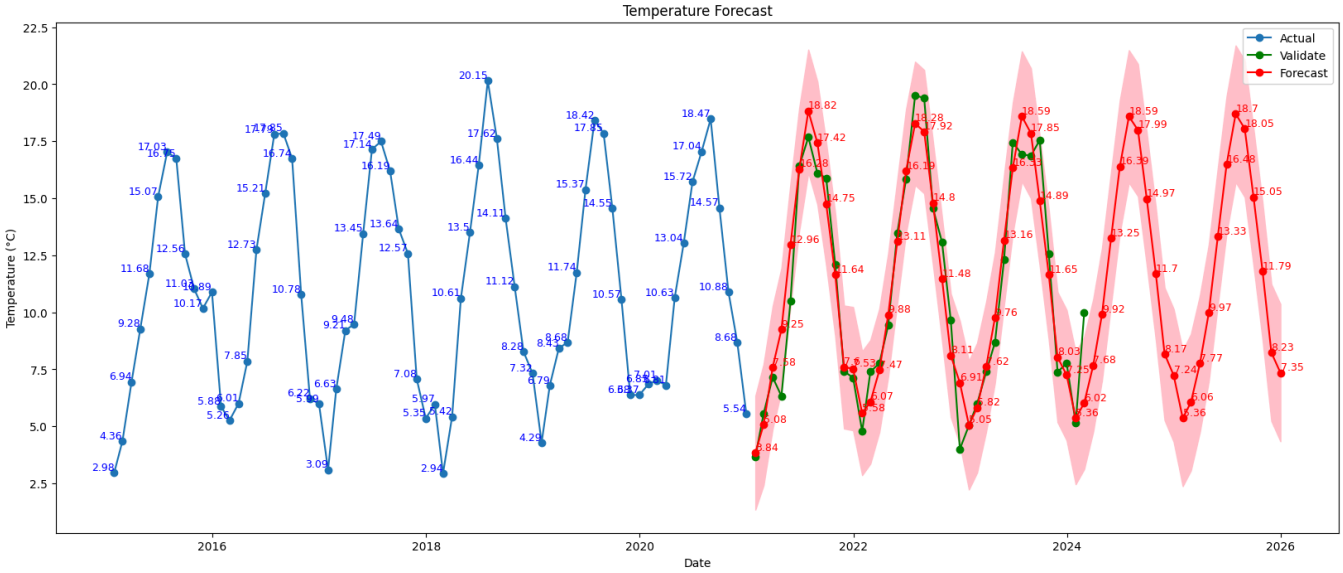
```
plot_acf(full_sarima_df_train)
plot_pacf(full_sarima_df_train)
plt.show()
```






```
p, d, q = 1, 1, 1
P, D, Q, s = 1, 1, 1, 12

# Fit the SARIMA model
model_default = SARIMAX(full_sarima_df_train, order=(p, d, q), seasonal_order=(P, D, Q, s))
results_default = model_default.fit()
```

```
# Forecast future values
forecast_periods_default = 60  # Forecast the next 60 months
forecast_default = results_default.get_forecast(steps=forecast_periods_default)
forecast_mean_default = forecast_default.predicted_mean
forecast_ci_default = forecast_default.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(full_sarima_df_train.index, full_sarima_df_train['Temp'], label='Actual', marker='o')
plt.plot(full_sarima_df_test.index, full_sarima_df_test['Temp'], label='Validate', color='Green', marker='o')
plt.plot(forecast_mean_default.index, forecast_mean_default, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_ci_default.index, forecast_ci_default.iloc[:, 0], forecast_ci_default.iloc[:, 1], color='pink')
plt.title("Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()

# Plot data points with their values
for i, value in enumerate(full_sarima_df_train['Temp']):
    plt.text(full_sarima_df_train.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# Plot data points with their values
for i, value in enumerate(forecast_mean_default):
    plt.text(forecast_mean_default.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()

# Calculate R-squared
r2 = r2_score(full_sarima_df_train[-forecast_periods_default:], forecast_mean_default)

# Calculate Mean Squared Error
mse = mean_squared_error(full_sarima_df_train[-forecast_periods_default:], forecast_mean_default)

# Calculate Root Mean Squared Error
rmse = mean_squared_error(full_sarima_df_train[-forecast_periods_default:], forecast_mean_default, squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(full_sarima_df_train[-forecast_periods_default:], forecast_mean_default)

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```

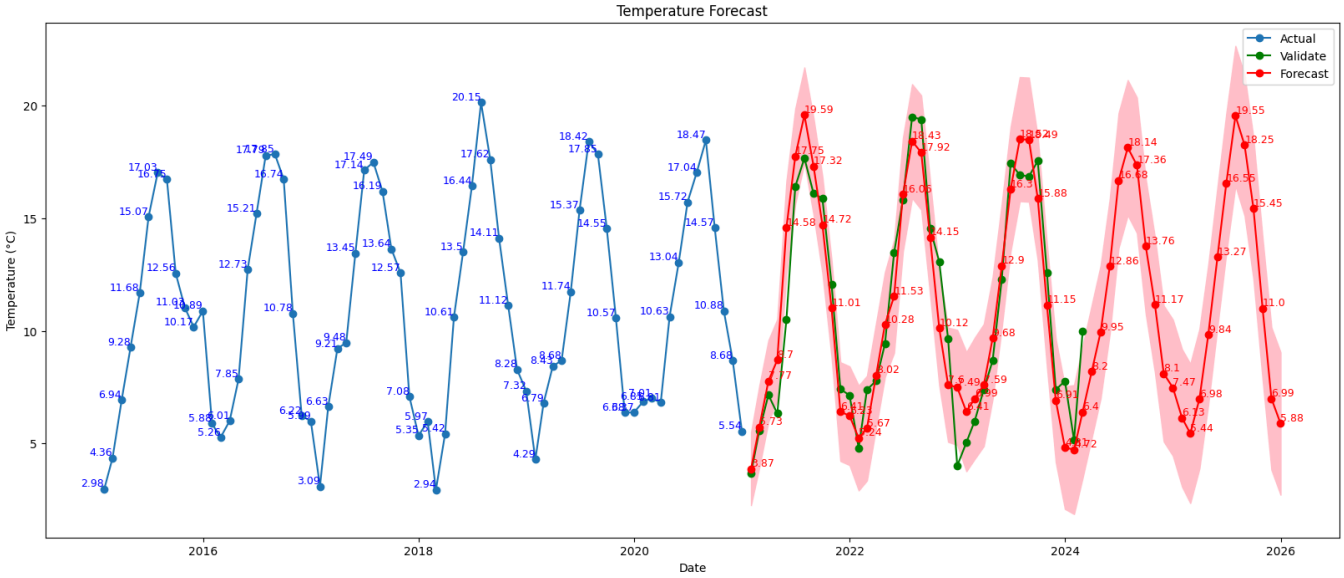



```
R-squared: 0.9370
Mean Squared Error: 1.40
Root Mean Squared Error: 1.18
Mean Absolute Error: 1.00
```

```
p, d, q = 3, 1, 8
P, D, Q, s = 3, 1, 8, 12

# Fit the SARIMA model
model = SARIMAX(full_sarima_df_train, order=(p, d, q), seasonal_order=(P, D, Q, s))
results = model.fit()
```

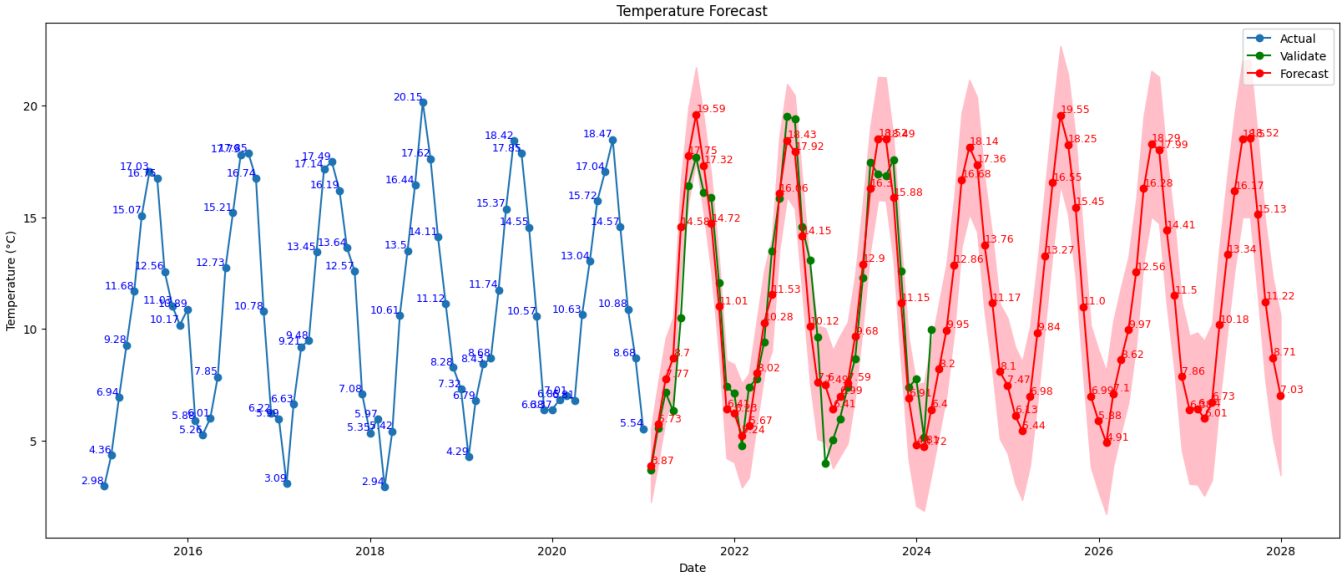
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate sta
  warn('Too few observations to estimate starting parameters%s.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
# Forecast future values
forecast_periods = 60  # Forecast the next 60 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(full_sarima_df_train.index, full_sarima_df_train['Temp'], label='Actual', marker='o')
plt.plot(full_sarima_df_test.index, full_sarima_df_test['Temp'], label='Validate', color='Green', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.title("Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()

# Plot data points with their values
for i, value in enumerate(full_sarima_df_train['Temp']):
    plt.text(full_sarima_df_train.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# Plot data points with their values
for i, value in enumerate(forecast_mean):
    plt.text(forecast_mean.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()

# Calculate R-squared
r2 = r2_score(full_sarima_df_train[-forecast_periods:], forecast_mean)

# Calculate Mean Squared Error
mse = mean_squared_error(full_sarima_df_train[-forecast_periods:], forecast_mean)

# Calculate Root Mean Squared Error
rmse = mean_squared_error(full_sarima_df_train[-forecast_periods:], forecast_mean, squared=False)

# Calculate Mean Absolute Error
mae = mean_absolute_error(full_sarima_df_train[-forecast_periods:], forecast_mean)

# Print the calculated metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```

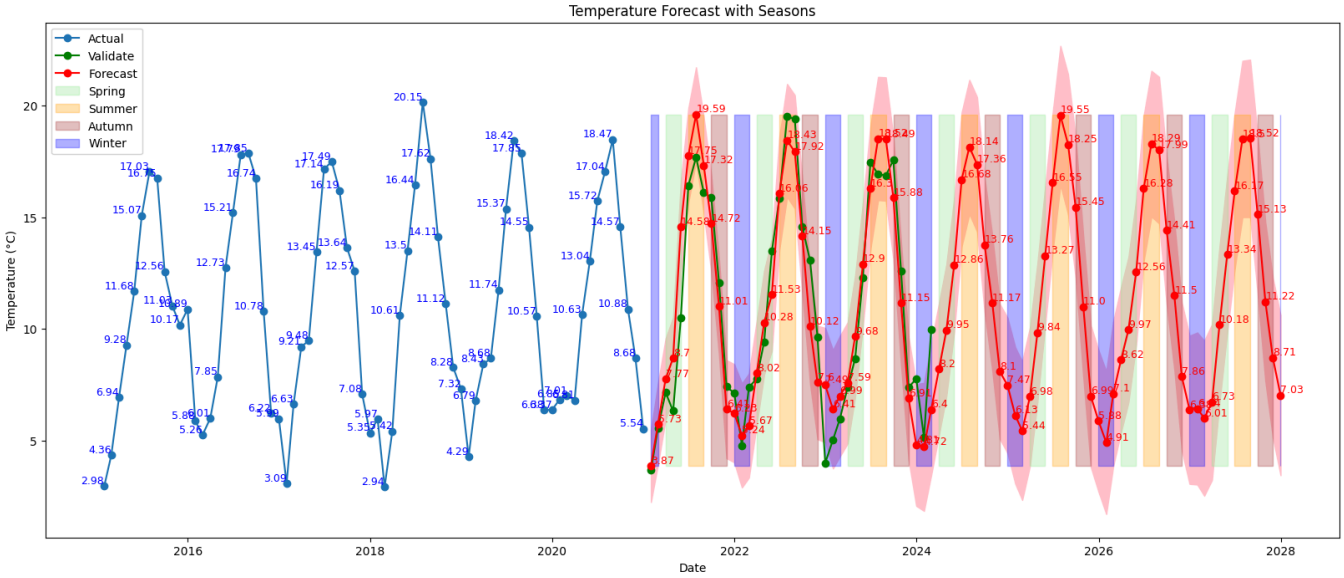



```
R-squared: 0.9133
Mean Squared Error: 1.92
Root Mean Squared Error: 1.39
Mean Absolute Error: 1.12
```

```
# Forecast future values
forecast_periods = 84  # Forecast the next 60 months
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(full_sarima_df_train.index, full_sarima_df_train['Temp'], label='Actual', marker='o')
plt.plot(full_sarima_df_test.index, full_sarima_df_test['Temp'], label='Validate', color='Green', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.title("Temperature Forecast")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()

# Plot data points with their values
for i, value in enumerate(full_sarima_df_train['Temp']):
    plt.text(full_sarima_df_train.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# Plot data points with their values
for i, value in enumerate(forecast_mean):
    plt.text(forecast_mean.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```




```
# Define seasons based on months
spring = ((forecast_mean.index.month >= 3) & (forecast_mean.index.month <= 5))
summer = ((forecast_mean.index.month >= 6) & (forecast_mean.index.month <= 8))
autumn = ((forecast_mean.index.month >= 9) & (forecast_mean.index.month <= 11))
winter = ((forecast_mean.index.month == 12) | (forecast_mean.index.month <= 2))

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(full_sarima_df_train.index, full_sarima_df_train['Temp'], label='Actual', marker='o')
plt.plot(full_sarima_df_test.index, full_sarima_df_test['Temp'], label='Validate', color='Green', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')

# Shade different seasons
plt.fill_between(forecast_mean.index, forecast_mean.min(), forecast_mean.max(), where=spring, color='lightgreen', alpha=0.3, label='Sprin
plt.fill_between(forecast_mean.index, forecast_mean.min(), forecast_mean.max(), where=summer, color='orange', alpha=0.3, label='Summer')
plt.fill_between(forecast_mean.index, forecast_mean.min(), forecast_mean.max(), where=autumn, color='brown', alpha=0.3, label='Autumn')
plt.fill_between(forecast_mean.index, forecast_mean.min(), forecast_mean.max(), where=winter, color='blue', alpha=0.3, label='Winter')

plt.title("Temperature Forecast with Seasons")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()

# Plot data points with their values
for i, value in enumerate(full_sarima_df_train['Temp']):
    plt.text(full_sarima_df_train.index[i], value, str(round(value, 2)), fontsize=9, ha='right', va='bottom', color='blue')

# Plot data points with their values
for i, value in enumerate(forecast_mean):
    plt.text(forecast_mean.index[i], value, str(round(value, 2)), fontsize=9, ha='left', va='bottom', color='red')

plt.show()
```




```
forecast_df = pd.DataFrame({'date': forecast_mean.index, 'predicted_temperature': forecast_mean.values})

# Display the dataframe
print(forecast_df)

forecast_df.to_excel('forecast.xlsx', index=False)
```

```
          date  predicted_temperature
0   2021-01-31               3.868536
1   2021-02-28               5.729180
2   2021-03-31               7.769748
3   2021-04-30               8.703348
4   2021-05-31              14.575274
..         ...                    ...
79  2027-08-31              18.522572
80  2027-09-30              15.126265
81  2027-10-31              11.219137
82  2027-11-30               8.707105
83  2027-12-31               7.027626

[84 rows x 2 columns]
```

```
# Forecast future values
forecast_periods = 60
forecast = results.get_forecast(steps=forecast_periods)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Find maximum and minimum values for observed data
max_observed_value_date = full_sarima_df_train['Temp'].idxmax()
max_observed_value = full_sarima_df_train['Temp'].max()
min_observed_value_date = full_sarima_df_train['Temp'].idxmin()
min_observed_value = full_sarima_df_train['Temp'].min()

# Find maximum and minimum values for forecasted data
max_forecast_value_date = forecast_mean.idxmax()
max_forecast_value = forecast_mean.max()
min_forecast_value_date = forecast_mean.idxmin()
min_forecast_value = forecast_mean.min()

# Plot the forecast
plt.figure(figsize=(20, 8))
plt.plot(full_sarima_df_train.index, full_sarima_df_train['Temp'], label='Actual', marker='o')
plt.plot(full_sarima_df_test.index, full_sarima_df_test['Temp'], label='Validate', color='Green', marker='o')
plt.plot(forecast_mean.index, forecast_mean, label='Forecast', color='red', marker='o')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink')
plt.title("Temperature Forecast")
```