EX 8 CREATE AN ARIMA MODEL FOR TIME SERIES FORECASTING

AIM:

To implement an ARIMA model for forecasting future daily average temperatures based on historical weather data.

ALGORITHM:

1. Load and preprocess the dataset, compute daily average temperature.
2. Check for stationarity using the ADF test and apply differencing if necessary.
3. Plot ACF and PACF to identify the ARIMA model parameters (p, d, q).
4. Fit the ARIMA model using the selected parameters.
5. Forecast the next 30 days and visualize the results.

PROGRAM:

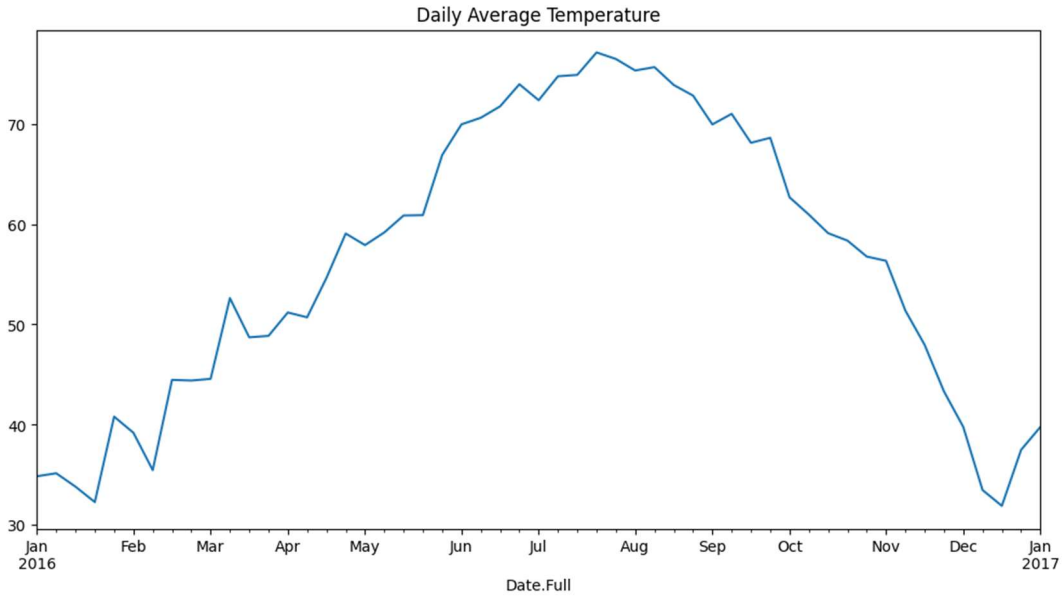
1. Import Necessary Libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

2. Load and Preprocess the Dataset:

```
# Load the dataset (ensure the path is correct)
data = pd.read_csv('weather.csv')
```

```
# Convert the 'Date.Full' column to datetime

data['Date.Full'] = pd.to_datetime(data['Date.Full'])

# Set the 'Date.Full' as the index

data.set_index('Date.Full', inplace=True)

# Group by date and calculate daily average temperature

daily_avg_temp = data.groupby('Date.Full')['Data.Temperature.Avg
Temp'].mean()

# Plot the daily average temperature to visualize the series

daily_avg_temp.plot(figsize=(12, 6))

plt.title('Daily Average Temperature')

plt.show()
```



3. Check for Stationarity:

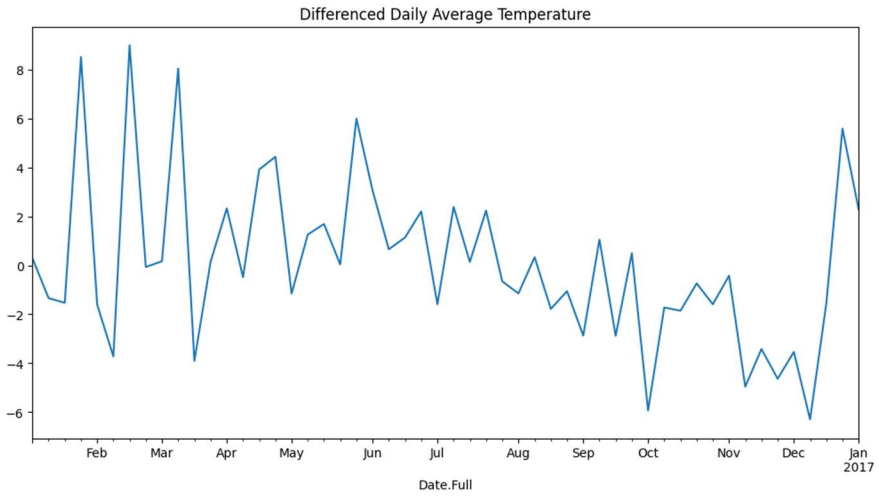
```
# Perform the ADF Test for stationarity

result = adfuller(daily_avg_temp.dropna())  # Drop NA values before running the
test
```

```
print('ADF Statistic:', result[0])
print('p-value:', result[1])
# If p-value < 0.05, the series is stationary; otherwise, we need differencing
```

```
ADF Statistic: -2.8710605184360745
p-value: 0.048818245821697505
```

4. Difference the Data if Necessary:

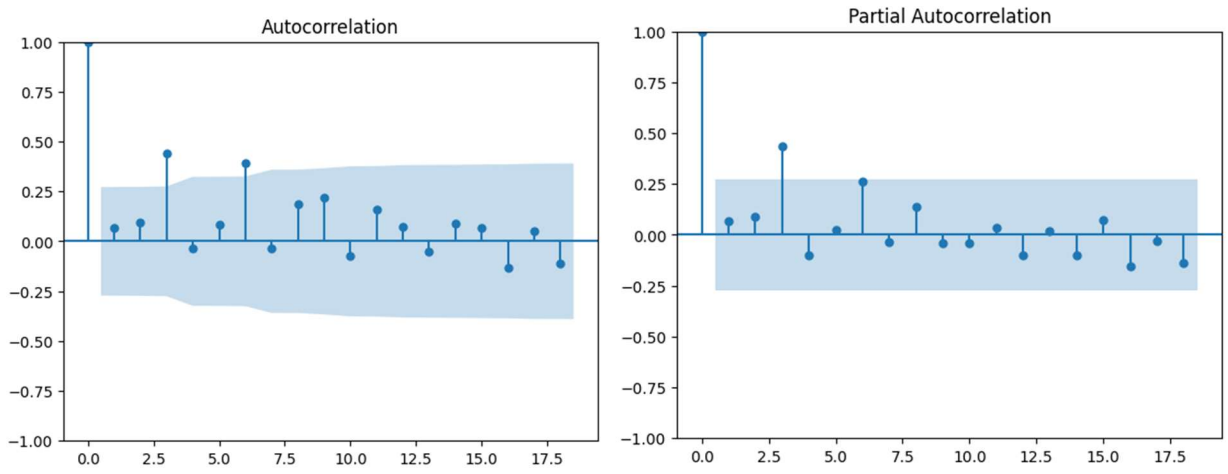
```
# If the series is not stationary (p-value > 0.05), apply differencing
daily_avg_temp_diff = daily_avg_temp.diff().dropna()

# Plot the differenced series
daily_avg_temp_diff.plot(figsize=(12, 6))
plt.title('Differenced Daily Average Temperature')
plt.show()
```



5. Plot ACF and PACF for Model Order Selection:

```
# Plot ACF and PACF to determine p and q (AR and MA components)
```

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



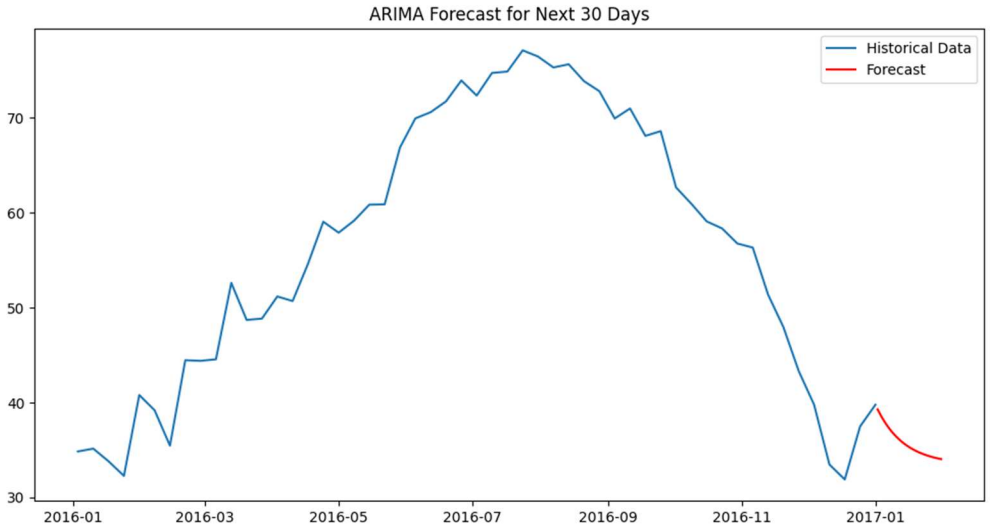
6. Fit the ARIMA Model:

```
# Fit ARIMA model (p=1, d=1, q=1) based on ACF and PACF analysis
model = ARIMA(daily_avg_temp, order=(1, 1, 1))  # ARIMA(p, d, q)
model_fit = model.fit()  # Ensure this line is executed
```

7. Forecast the Next 30 Days:

```
# Ensure that the model has been fitted before proceeding with forecasting
forecast_steps = 30
forecast = model_fit.forecast(steps=forecast_steps)
# Plot the forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_avg_temp, label='Historical Data')
```

```
plt.plot(pd.date_range(daily_avg_temp.index[-1], periods=forecast_steps + 1,
freq='D')[1:], forecast, label='Forecast', color='red')

plt.title(f'ARIMA Forecast for Next {forecast_steps} Days')

plt.legend()

plt.show()
```



RESULT:

The ARIMA model successfully forecasted the daily average temperatures for the next 30 days, capturing the expected trends and seasonality in the data. The forecasted values provided valuable insights into future temperature patterns.