Mt Washington Daily Temperature

About Dataset

This dataset contains time series data of daily temperature of washington from year 2014 to 2018.

Objective

The objective of this project is to analyse trends in Washington Daily Temperature data and build a forecasting model for the same.

Importing all the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Data Collection (CSV File)

In [2]:
```
df = pd.read_csv("MtWashingtonDailyTemps.csv")
df.head()
```

Out[2]:

	DATE	MinTemp	MaxTemp	AvgTemp	AvgWindSpeed	Sunrise	Sunset
0	12/1/2014	3	36	20	65.1	700	1608
1	12/2/2014	1	22	12	34.7	702	1607
2	12/3/2014	8	32	20	53.0	703	1607
3	12/4/2014	-5	9	2	60.2	704	1607
4	12/5/2014	6	17	12	30.5	705	1607

Data Preprocessing

Checking for datatypes of columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   DATE          1461 non-null   object
 1   MinTemp       1461 non-null   int64
 2   MaxTemp       1461 non-null   int64
 3   AvgTemp       1461 non-null   int64
 4   AvgWindSpeed  1461 non-null   float64
 5   Sunrise       1461 non-null   int64
 6   Sunset        1461 non-null   int64
dtypes: float64(1), int64(5), object(1)
memory usage: 80.0+ KB
```

convert the date column as index.

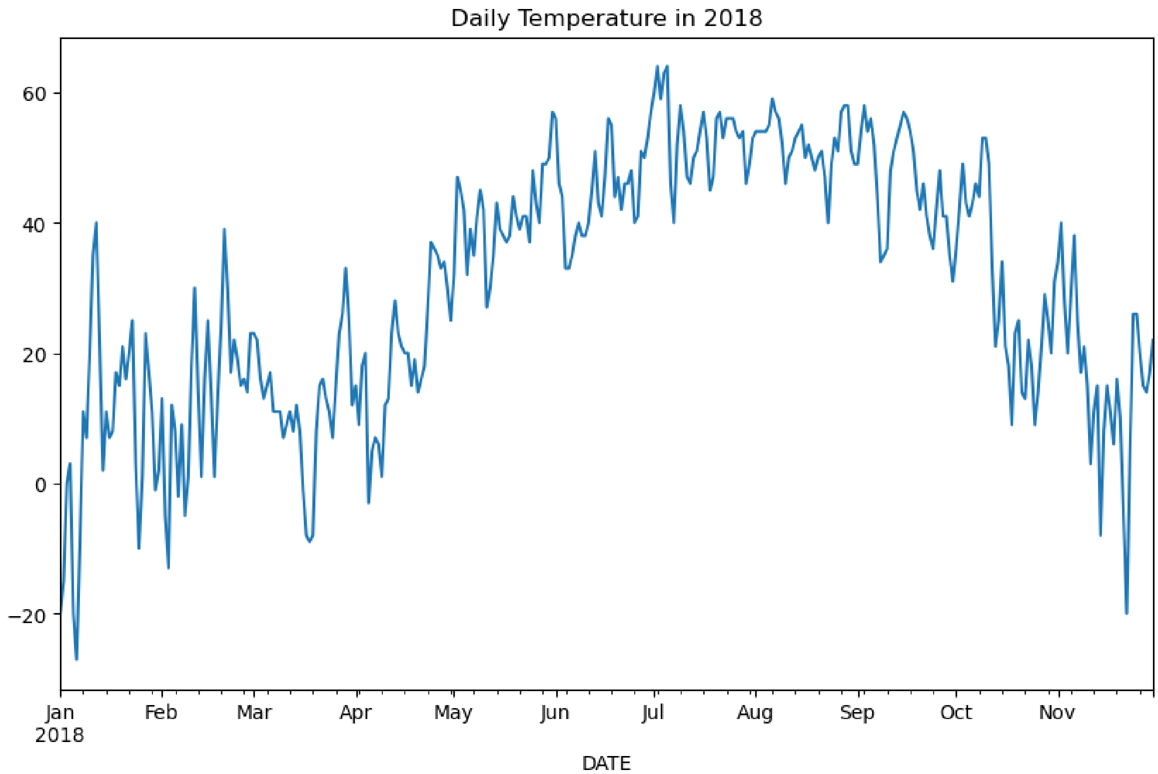
In [4]:
```
df=pd.read_csv("MtWashingtonDailyTemps.csv",parse_dates=True,index_col="DATE"
df.head()
```

Out[4]:

DATE	MinTemp	MaxTemp	AvgTemp	AvgWindSpeed	Sunrise	Sunset
2014-12-01	3	36	20	65.1	700	1608
2014-12-02	1	22	12	34.7	702	1607
2014-12-03	8	32	20	53.0	703	1607
2014-12-04	-5	9	2	60.2	704	1607
2014-12-05	6	17	12	30.5	705	1607

In [5]:
```
df['2018'].AvgTemp.plot(figsize=(10,6))
plt.title('Daily Temperature in 2018')
```

Out[5]:
```
Text(0.5, 1.0, 'Daily Temperature in 2018')
```



In [6]:
```
#Set the freq as "D"
df.index.freq='D'
```

In [7]:
```
df.index
```

Out[7]:
```
DatetimeIndex(['2014-12-01', '2014-12-02', '2014-12-03', '2014-12-04',
               '2014-12-05', '2014-12-06', '2014-12-07', '2014-12-08',
               '2014-12-09', '2014-12-10',
               ...
               '2018-11-21', '2018-11-22', '2018-11-23', '2018-11-24',
               '2018-11-25', '2018-11-26', '2018-11-27', '2018-11-28',
               '2018-11-29', '2018-11-30'],
              dtype='datetime64[ns]', name='DATE', length=1461, freq='D')
```

Dropping unnecessary columns

In [8]:
```
df=pd.DataFrame(df["AvgTemp"])
df.head()
```

Out[8]:

	AvgTemp
DATE	
2014-12-01	20
2014-12-02	12
2014-12-03	20
2014-12-04	2
2014-12-05	12

Stationary Check

check the given data is stationary or non-stationary using Augmented Dicky-Fuller Test.

In [9]:
```
from statsmodels.tsa.stattools import adfuller
print("Dicky-Fuller Result")
result=adfuller(df["AvgTemp"])
result
```

```
Dicky-Fuller Result
```

Out[9]:
```
(-2.2550045607147746,
 0.18689059009935127,
 19,
 1441,
 {'1%': -3.4348961395618476,
  '5%': -2.863547812296987,
  '10%': -2.5678389447194556},
 9699.403392324713)
```

The p-value is greater than 0.05 hence we don't reject null hypothesis. This means our data is non-stationary.

In [10]:
```
df1=pd.Series(result[0:4],index=["Adf test static","P-value","# lags used","#(
df1
```

Out[10]:
```
Adf test static      -2.255005
P-value               0.186891
# lags used          19.000000
#Observations      1441.000000
dtype: float64
```

In [11]:
```
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')
    result = adfuller(timeseries, autolag = 'AIC')
    result = pd.Series(result[0:4], index = ['Test Statistic','p-value','No. 
                                            'Number of Observations Used'])

    print (result)

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")


adf_test(df)
```

```
Results of Dickey-Fuller Test:
Test Statistic                   -2.255005
p-value                           0.186891
No. of Lags Used                 19.000000
Number of Observations Used    1441.000000
dtype: float64
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

The p-value is greater than 0.05 hence we to differencing the data to making it stationary.

Data Transformation

In [12]:
```
df["d1"]=df["AvgTemp"].diff()
df
```

Out[12]:

DATE	AvgTemp	d1
2014-12-01	20	NaN
2014-12-02	12	-8.0
2014-12-03	20	8.0
2014-12-04	2	-18.0
2014-12-05	12	10.0
...	...	...
2018-11-26	20	-6.0
2018-11-27	15	-5.0
2018-11-28	14	-1.0
2018-11-29	17	3.0
2018-11-30	22	5.0

1461 rows × 2 columns

Drop null values

In [13]:
```
df.dropna(inplace=True)
df
```

Out[13]:

DATE	AvgTemp	d1
2014-12-02	12	-8.0
2014-12-03	20	8.0
2014-12-04	2	-18.0
2014-12-05	12	10.0
2014-12-06	20	8.0
...	...	...
2018-11-26	20	-6.0
2018-11-27	15	-5.0
2018-11-28	14	-1.0
2018-11-29	17	3.0
2018-11-30	22	5.0

1460 rows × 2 columns

In [14]:
```
print("Dicky-Fuller Result")
result1=adfuller(df["d1"])
result1
```

```
Dicky-Fuller Result
```

Out[14]:
```
(-12.931121457879936,
 3.6987943076411585e-24,
 18,
 1441,
 {'1%': -3.4348961395618476,
  '5%': -2.863547812296987,
  '10%': -2.5678389447194556},
 9696.198893776458)
```

In [15]:
```
df2=pd.Series(result1[0:4],index=["Adf test static","P-value","# lags used","
df2
```

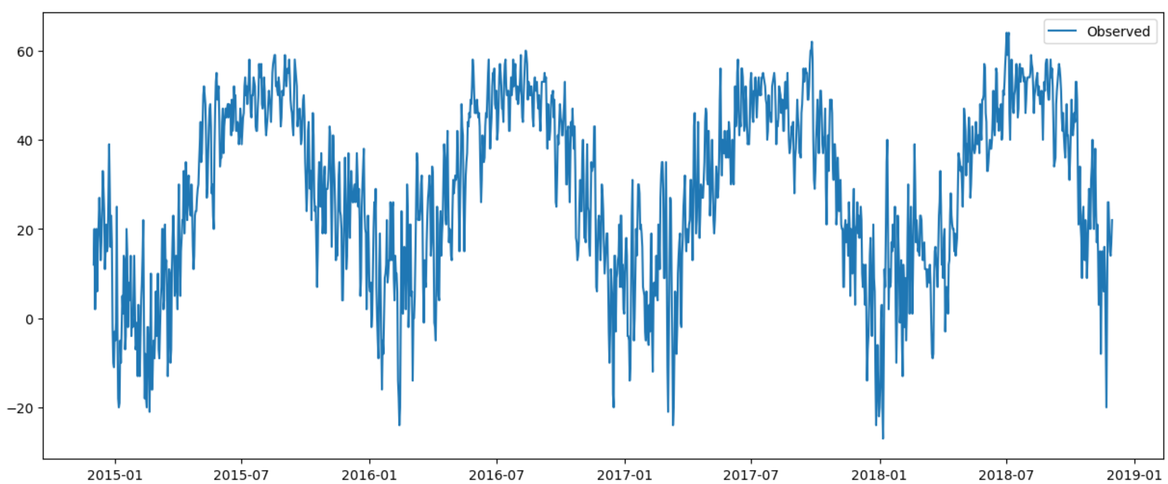
Out[15]:
```
Adf test static   -1.293112e+01
P-value            3.698794e-24
# lags used        1.800000e+01
#Observations      1.441000e+03
dtype: float64
```

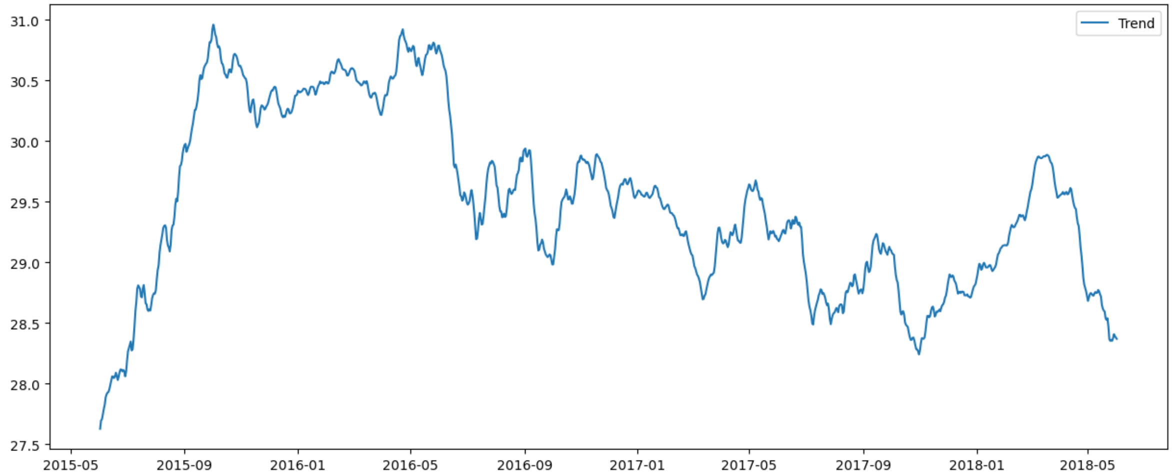
The p-value is lesser than 0.05 hence we reject null hypothesis. This means our data is stationary.

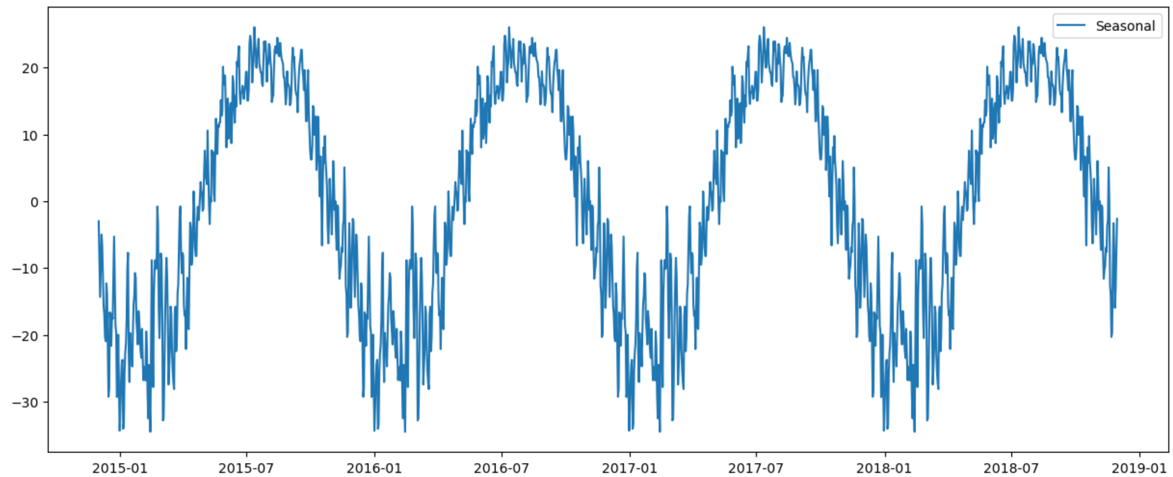
Decompose the time series into trend,seasonal and residual components.

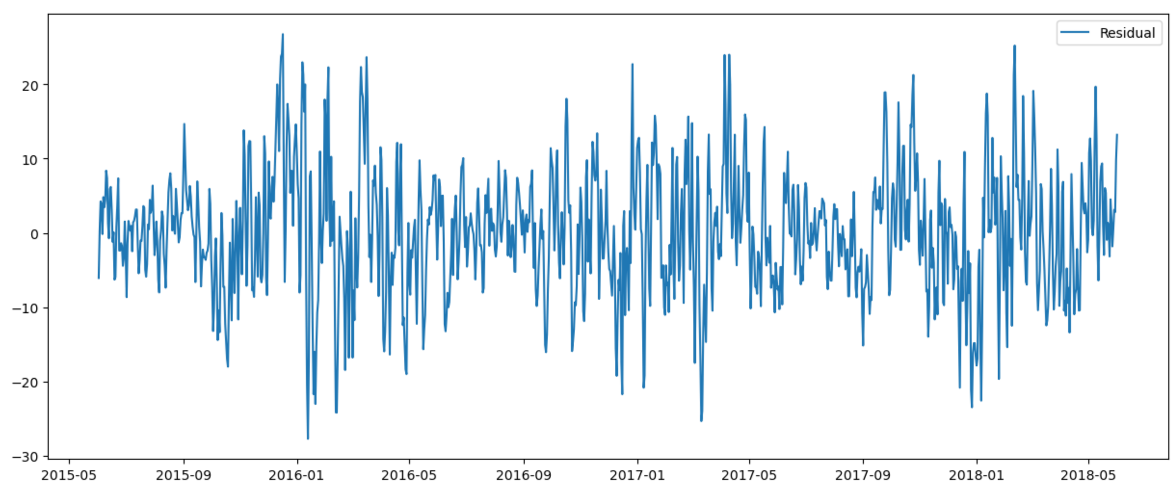
In [16]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition=seasonal_decompose(df["AvgTemp"],model="additive",period=365)

#Plotting the observed values:
observed=decomposition.observed
plt.figure(figsize=(15,6))
plt.plot(observed,label="Observed")
plt.legend()

#Plotting the trend component
trend=decomposition.trend
plt.figure(figsize=(15,6))
plt.plot(trend,label="Trend")
plt.legend()

#Plotting the seasonal component
seasonal=decomposition.seasonal
plt.figure(figsize=(15,6))
plt.plot(seasonal,label="Seasonal")
plt.legend()

#Plotting the residual component
residual=decomposition.resid
plt.figure(figsize=(15,6))
plt.plot(residual,label="Residual")
plt.legend()
```

Out[16]: <matplotlib.legend.Legend at 0x2263eba6d60>




Trend
31.0
30.5
30.0
29.5
29.0
28.5
28.0
27.5
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05


Seasonal
20
10
0
−10
−20
−30
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01


Residual
20
10
0
−10
−20
−30
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05

In [17]:
```
y=df["AvgTemp"]
y
```

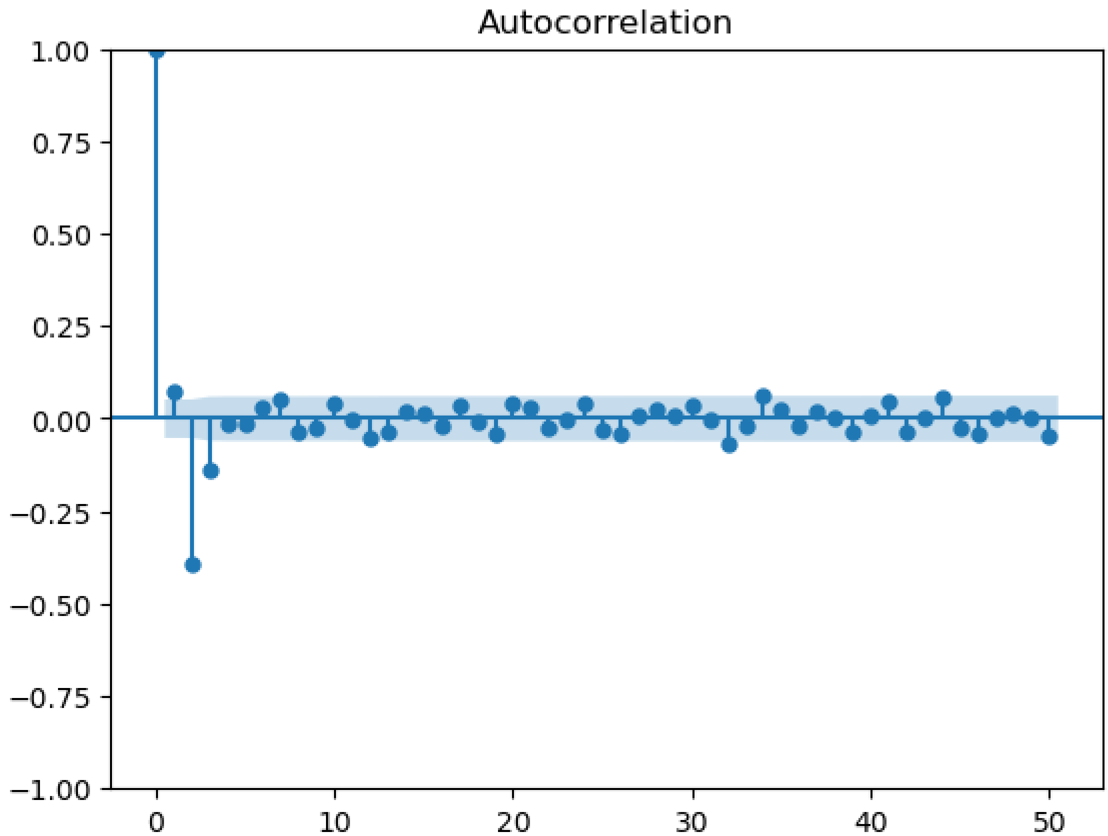
Out[17]:
```
DATE
2014-12-02    12
2014-12-03    20
2014-12-04     2
2014-12-05    12
2014-12-06    20
              ..
2018-11-26    20
2018-11-27    15
2018-11-28    14
2018-11-29    17
2018-11-30    22
Freq: D, Name: AvgTemp, Length: 1460, dtype: int64
```

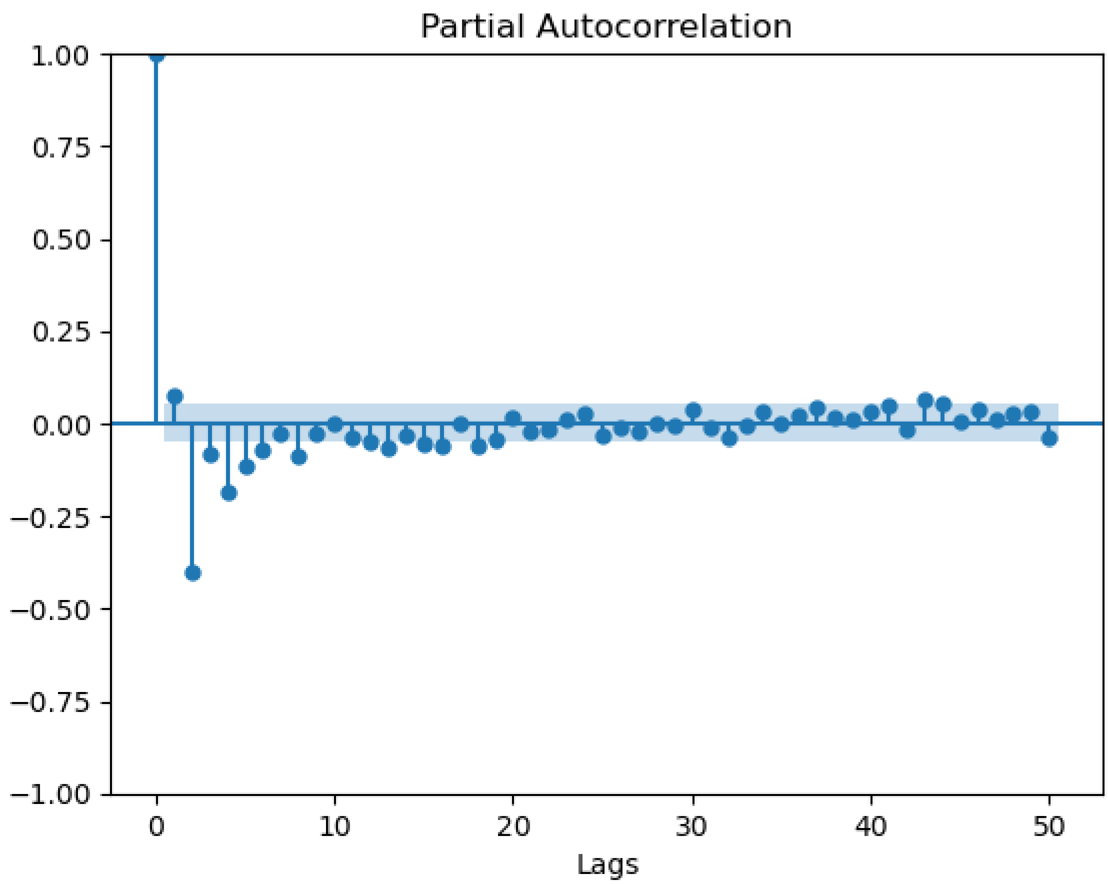
In [18]:
```
y["2018":]
```

Out[18]:
```
DATE
2018-01-01   -20
2018-01-02   -15
2018-01-03     0
2018-01-04     3
2018-01-05   -20
              ..
2018-11-26    20
2018-11-27    15
2018-11-28    14
2018-11-29    17
2018-11-30    22
Freq: D, Name: AvgTemp, Length: 334, dtype: int64
```

The Autoregressive Integrated Moving Average (ARIMA) Model:

In [19]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

plot_acf(df["d1"],lags=50)
plot_pacf(df["d1"],lags=50)
plt.xlabel("Lags")
plt.show()
```





Model Selection

Applying ARIMA model

In [20]:
```
from pmdarima.arima import auto_arima
model_arima = auto_arima(y, seasonal=False, stepwise=True)
order_arima = model_arima.order
```

In [21]:
```
order_arima
```

Out[21]: (0, 1, 3)

Model Fitting

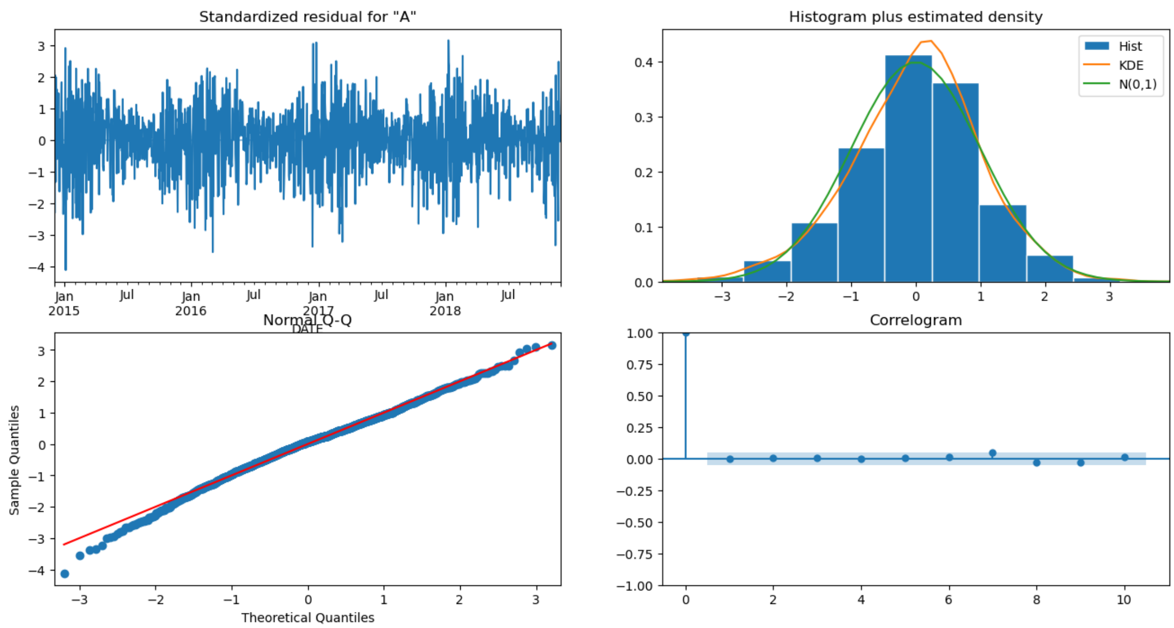
In [22]:
```
from statsmodels.tsa.arima.model import ARIMA
arima_model = ARIMA(y, order=(0,1,3),)
arima_fit = arima_model.fit()
arima_forecast = arima_fit.forecast(steps=30)  # Forecasting 30 steps ahead
```

In [23]:
```
mod = ARIMA(y,order=(0,1,3),)
results1 = mod.fit()
print(results1.summary().tables[1])
```

```
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ma.L1          0.0083      0.023      0.362      0.718      -0.036       0.0
53
ma.L2         -0.5193      0.019    -27.906      0.000      -0.556      -0.4
83
ma.L3         -0.1795      0.022     -8.120      0.000      -0.223      -0.1
36
sigma2        50.0353      1.665     30.043      0.000      46.771      53.3
00
==============================================================================
==
```

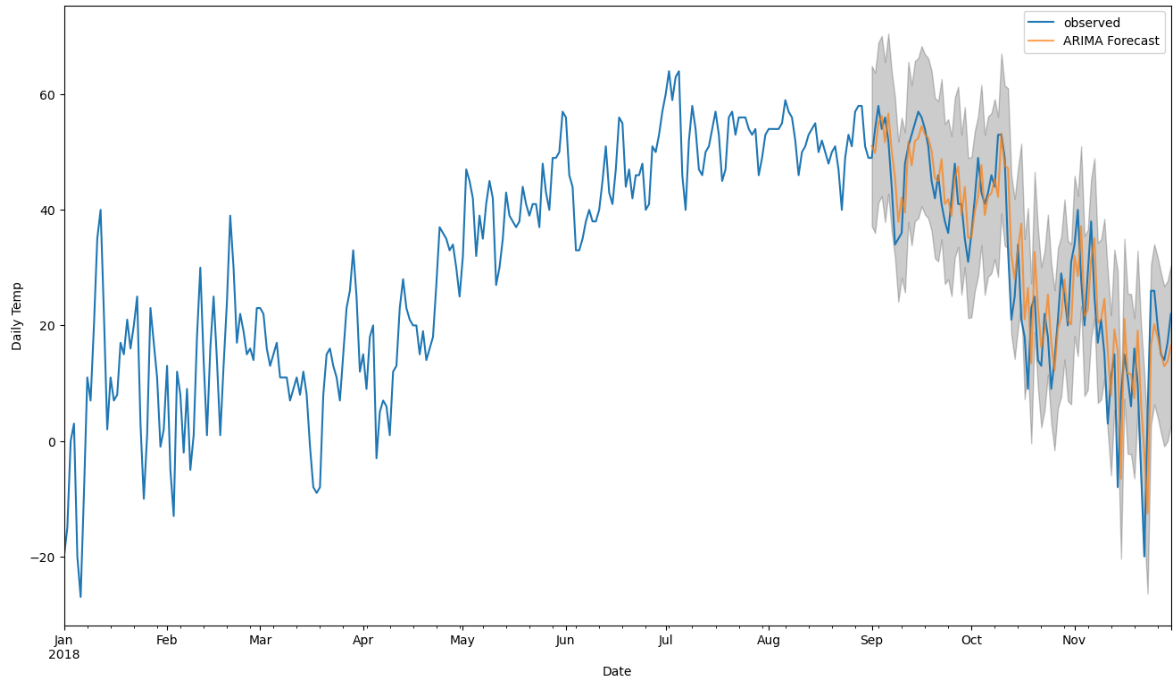
Plotting the Diagnostics Plot

In [24]:
```
results1.plot_diagnostics(figsize=(16, 8))
plt.show()
```

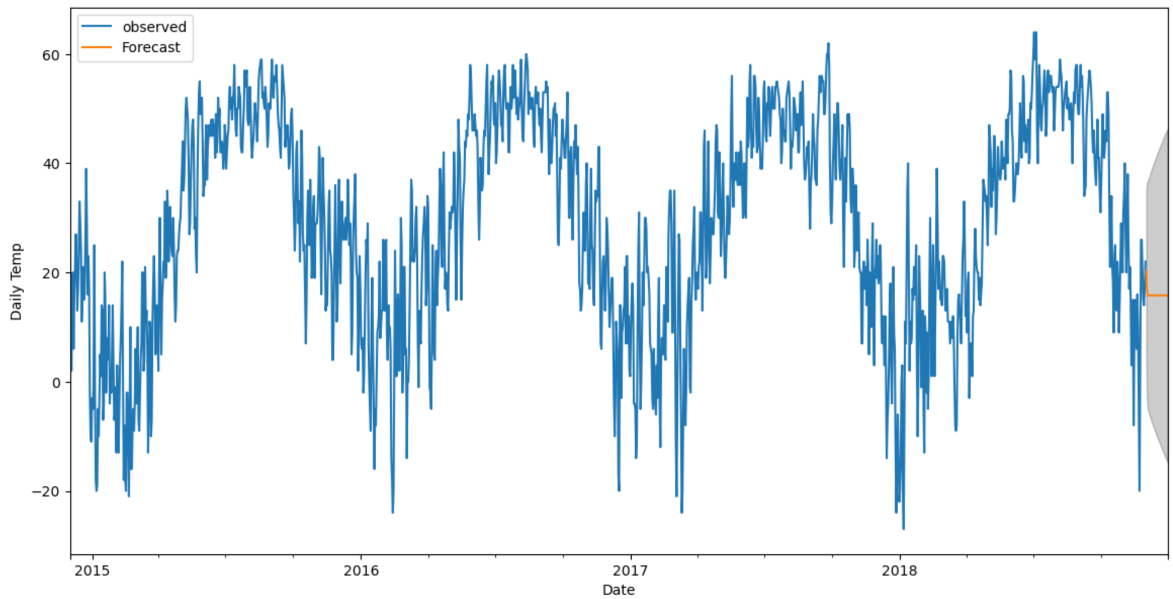


ARIMA Forecast

In [25]:
```
pred = results1.get_prediction(start=pd.to_datetime('2018-09-01'), dynamic=Fa
pred_ci = pred.conf_int()
ax = y["2018":].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='ARIMA Forecast', alpha=.7, figsize=(16
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Daily Temp')
plt.legend()
plt.show()
```



In [26]:
```
#Forecasting for next 30 days.

pred = results1.get_forecast(steps=30)
pred_ci = pred.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=0.2)
ax.set_xlabel('Date')
ax.set_ylabel('Daily Temp')
plt.legend()
plt.show()
```



SARIMAX

In [27]:
```
# Applying SARIMA model
from statsmodels.tsa.statespace.sarimax import SARIMAX

model_sarima = auto_arima(y, seasonal=True, stepwise=True, m=7)
order_sarima = model_sarima.order
seasonal_order_sarima = model_sarima.seasonal_order
```

Training the SARIMAX Model:

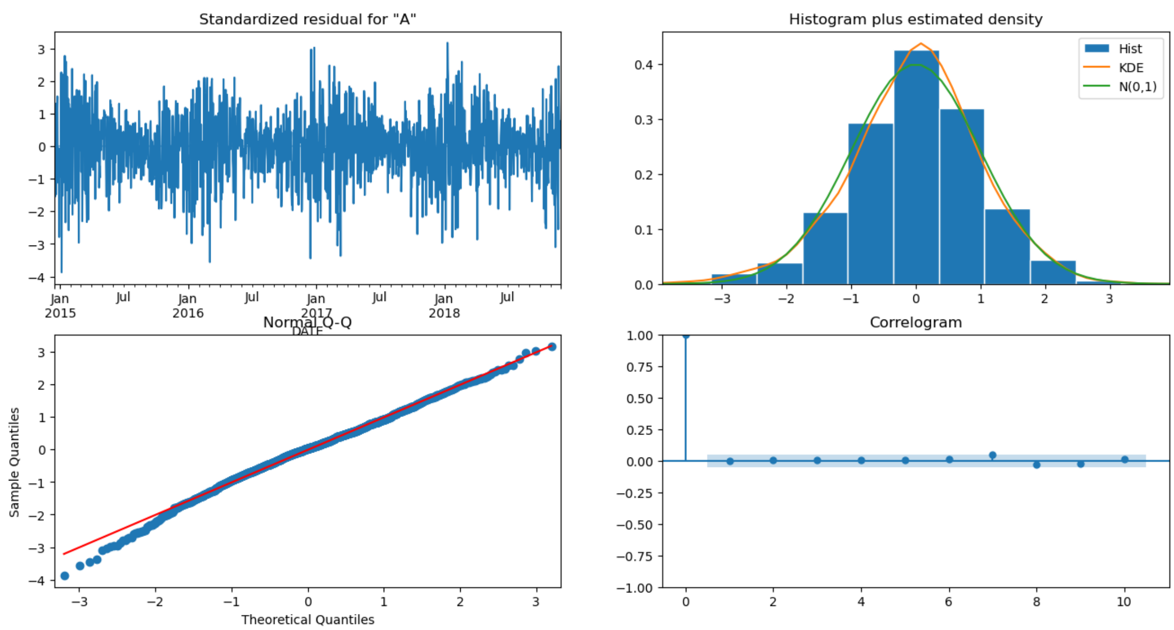
Now we train the SARIMAX model using the optimal values of the hyperparameters that we found using auto-ARIMA.

In [28]:
```
sarima_model = SARIMAX(df["AvgTemp"], order=order_sarima, seasonal_order=seas
sarima_fit = sarima_model.fit()
sarima_forecast = sarima_fit.forecast(steps=30)  # Forecasting 30 steps ahead
```

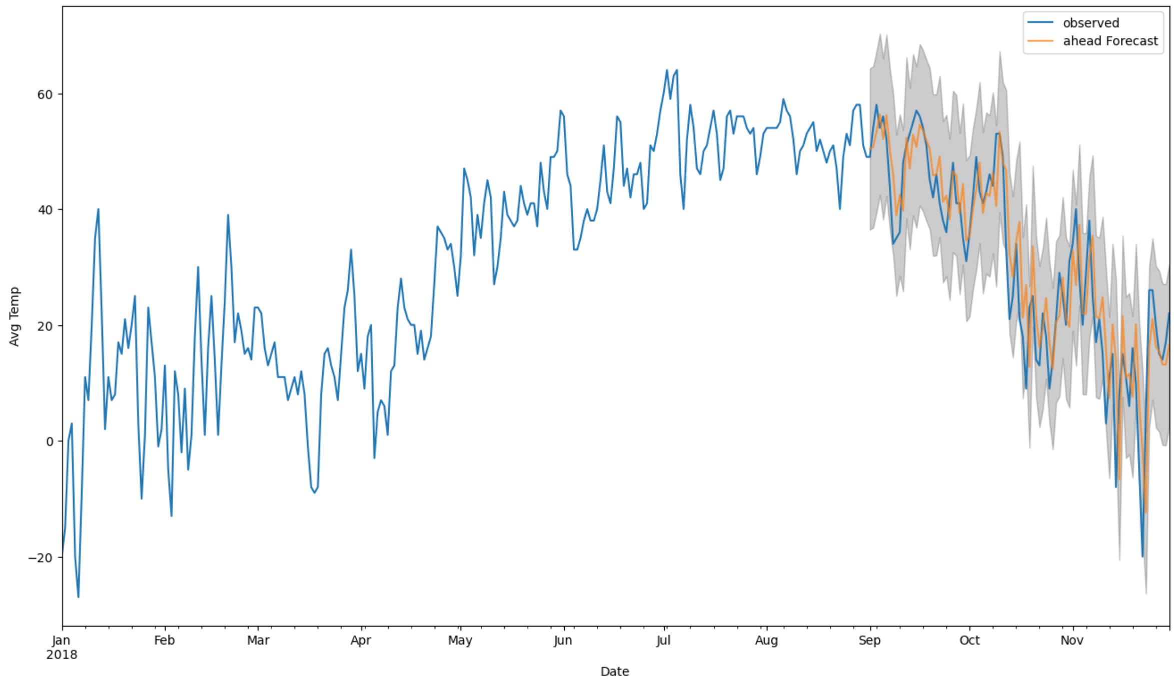
In [29]:
```
mod = SARIMAX(y,order=(0,1,3),seasonal_order=(0, 1, 1, 12),)
results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ma.L1          0.0120      0.023      0.517      0.605      -0.034       0.0
58
ma.L2         -0.5183      0.019    -27.483      0.000      -0.555      -0.4
81
ma.L3         -0.1815      0.023     -7.909      0.000      -0.226      -0.1
37
ma.S.L12      -0.9998      1.068     -0.937      0.349      -3.092       1.0
92
sigma2        50.0101     53.288      0.938      0.348     -54.433     154.4
53
==============================================================================
==
```

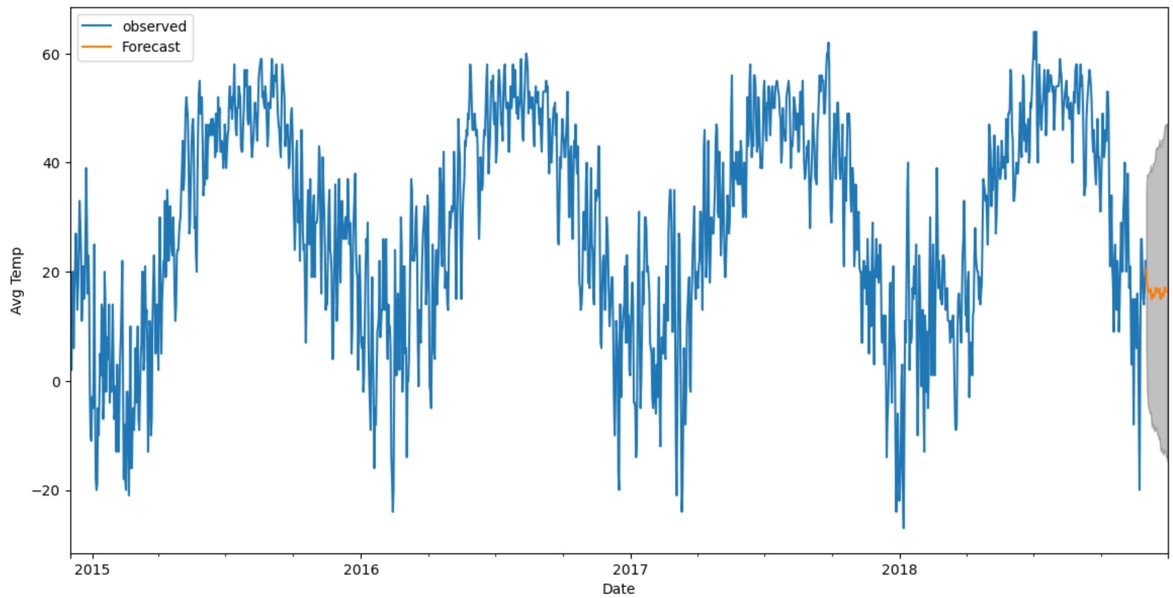
In [30]:
```
results.plot_diagnostics(figsize=(16, 8))
plt.show()
```



In [31]:
```
pred = results.get_prediction(start=pd.to_datetime('2018-09-01'), dynamic=Fal
pred_ci = pred.conf_int()
ax = y['2018':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='ahead Forecast', alpha=0.7, figsize=(1
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Avg Temp')
plt.legend()
plt.show()
```



In [32]:
```
pred1 = results.get_forecast(steps=30)
pred_ci = pred1.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred1.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Avg Temp')
plt.legend()
plt.show()
```



In [33]:
```
y_forecasted = pred.predicted_mean
y_truth = y['2018-09-01':]
mse = ((y_forecasted - y_truth) ** 2).mean()
```

Model Evaluation

In [34]:
```
print('The Mean Squared Error (MSE) of our forecasts is {}'.format(round(mse,
print('The Root Mean Squared Error (RMSE) of our forecasts is {}'.format(roun
```

```
The Mean Squared Error (MSE) of our forecasts is 56.9
The Root Mean Squared Error (RMSE) of our forecasts is 7.54
```

LSTM

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

In [35]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Convert the DataFrame to a NumPy array
data = df['AvgTemp'].values
data = data.reshape(-1, 1)  # Convert to a 2D array (required for MinMaxScale
```

In [36]:
```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

In [37]:
```
# Split the data into training and testing sets
train_size = int(len(data) * 0.8) #multiplying by 8 because the data is split
#80% data will be the training data
#20% data will be the testing data
train= data[:train_size]
test= data[train_size:]
```

In [38]:
```
# Create sequences and labels for the LSTM model
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)
```

In [39]:
```
seq_length = 7  # Length of input sequence
X_train, y_train = create_sequences(train, seq_length)
X_test, y_test = create_sequences(test, seq_length)
```

In [40]:
```
# Reshape the data for LSTM input (samples, time steps, features)
X_train = X_train.reshape(X_train.shape[0], seq_length, 1)
X_test = X_test.reshape(X_test.shape[0], seq_length, 1)
```

In [41]:
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(seq_length, 1)))
#Here, seq_length represents the length of each input sequence, and 1 indicate
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=1)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0063
Epoch 78/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 79/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0063
Epoch 80/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0063
Epoch 81/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 82/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0063
Epoch 83/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 84/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 85/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 86/100
1161/1161 [==============================] - 6s 5ms/step - loss: 0.0062
Epoch 87/100
```

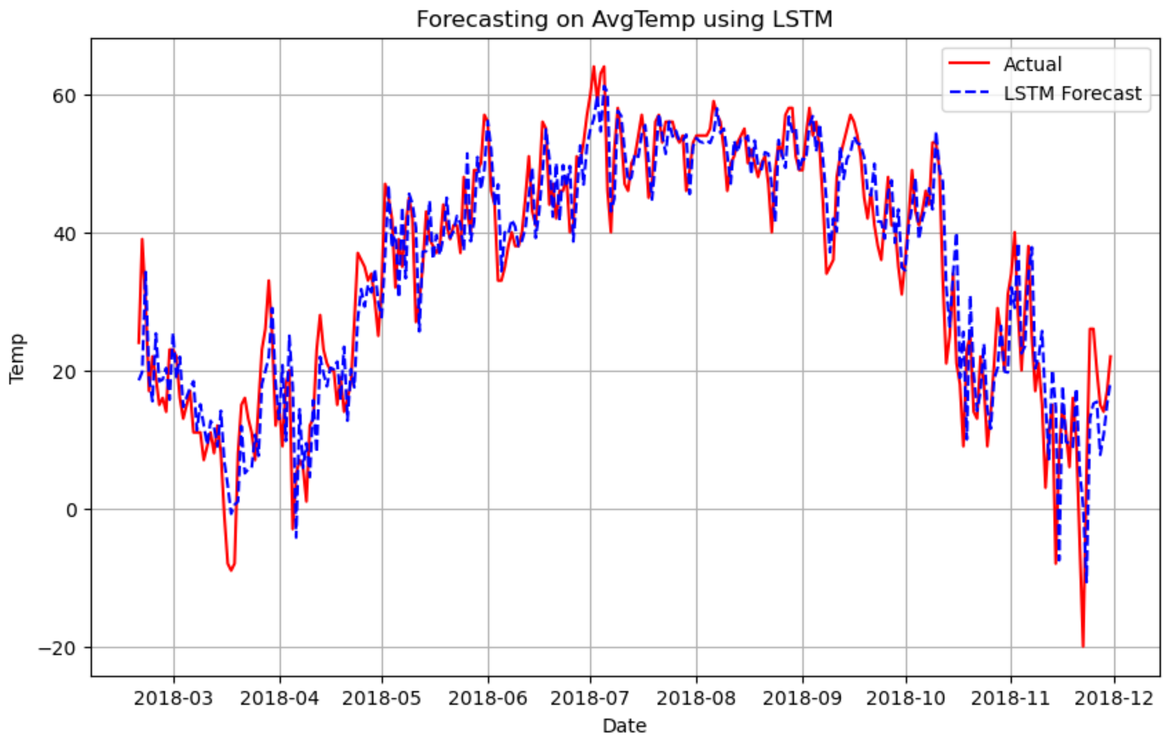
In [42]:
```
# Inverse transform the predictions and actual values to the original scale
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)
```

In [43]:
```
# Calculate RMSE for LSTM
rmse_lstm = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
print(f"LSTM RMSE: {rmse_lstm}")
```

```
LSTM RMSE: 6.343664131321569
```

Forecasting

In [44]:
```
# Plot the forecasts
plt.figure(figsize=(10, 6))
plt.plot(df.index[train_size+seq_length:], y_test_inv, label='Actual', color=
plt.plot(df.index[train_size+seq_length:], y_pred_inv, label='LSTM Forecast',
plt.xlabel('Date')
plt.ylabel('Temp')
plt.title('Forecasting on AvgTemp using LSTM')
plt.legend()
plt.grid(True)
plt.show()
```



In [45]:
```
print("RMSE for SARIMAX Model : 7.54")
print("RMSE for LSTM Model : 6.34")
```

```
RMSE for SARIMAX Model : 7.54
RMSE for LSTM Model : 6.34
```

LSTM model perform well on this time series dataset.

In [ ]: