```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# 1. Load data
df = pd.read_csv("price_climate.csv", parse_dates=["date"])
df.set_index("date", inplace=True)

# 2. Select features
data = df[["Price_Monthly_Avg", "PRCP_Monthly_Avg", "TAVG_Monthly_Avg"]].fillna(0)

# 3. Normalize features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# 4. Create sequences
def create_sequences(data, seq_len):
    X, y = [], []
    for i in range(seq_len, len(data)):
        X.append(data[i-seq_len:i])
        y.append(data[i, 0])  # predict price only
    return np.array(X), np.array(y)

seq_len = 18  # use past 12 months
X, y = create_sequences(scaled_data, seq_len)



split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

print(f"Test size: {len(y_test)}")
print(len(X))
print(len(X)*0.8)
print(len(df))
```

```
Test size: 69
344
275.2
362
```

```
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(seq_len, X.shape[2])),
    LSTM(32),
    Dense(1)
])

model.compile(loss='mse', optimizer='adam')
model.summary()

# Train
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
                    validation_data=(X_test, y_test), verbose=1)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_di
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 18, 64)	17,408
lstm_1 (LSTM)	(None, 32)	12,416
dense (Dense)	(None, 1)	33

```
 Total params: 29,857 (116.63 KB)
 Trainable params: 29,857 (116.63 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - loss: 0.0328 - val_loss: 0.0581
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0058 - val_loss: 0.0366
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0034 - val_loss: 0.0327
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 0.0025 - val_loss: 0.0286
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0018 - val_loss: 0.0231
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0018 - val_loss: 0.0171
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0013 - val_loss: 0.0223
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0015 - val_loss: 0.0165
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0014 - val_loss: 0.0252
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0017 - val_loss: 0.0165
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0012 - val_loss: 0.0156
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0012 - val_loss: 0.0162
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0013 - val_loss: 0.0144
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0012 - val_loss: 0.0136
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 0.0010 - val_loss: 0.0165
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 8.9459e-04 - val_loss: 0.0153
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 9.5076e-04 - val_loss: 0.0125
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0011 - val_loss: 0.0166
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 0.0010 - val_loss: 0.0125
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 8.6797e-04 - val_loss: 0.0118
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 7.4098e-04 - val_loss: 0.0119
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 8.0242e-04 - val_loss: 0.0112
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 7.2798e-04 - val_loss: 0.0119
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 7.7406e-04 - val_loss: 0.0104
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 8.7132e-04 - val_loss: 0.0112
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 8.2406e-04 - val_loss: 0.0093
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 7.8549e-04 - val_loss: 0.0111
Epoch 28/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 8.6148e-04 - val_loss: 0.0104
Epoch 29/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 7.8763e-04 - val_loss: 0.0105
Epoch 30/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 7.8154e-04 - val_loss: 0.0084
Epoch 31/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 9.9405e-04 - val_loss: 0.0119
Epoch 32/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 9.0382e-04 - val_loss: 0.0102
Epoch 33/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 7.1706e-04 - val_loss: 0.0086
Epoch 34/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 6.3544e-04 - val_loss: 0.0097
Epoch 35/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 7.9841e-04 - val_loss: 0.0086
Epoch 36/50
```

```
Epoch 36/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 7.5090e-04 - val_loss: 0.0097
Epoch 37/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.2066e-04 - val_loss: 0.0080
Epoch 38/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0011 - val_loss: 0.0096
Epoch 39/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 8.2544e-04 - val_loss: 0.0084
Epoch 40/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 7.9886e-04 - val_loss: 0.0083
Epoch 41/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 6.9292e-04 - val_loss: 0.0093
Epoch 42/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 5.7037e-04 - val_loss: 0.0080
Epoch 43/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 6.0130e-04 - val_loss: 0.0080
Epoch 44/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.1359e-04 - val_loss: 0.0088
Epoch 45/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 5.4958e-04 - val_loss: 0.0078
Epoch 46/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 5.6154e-04 - val_loss: 0.0081
Epoch 47/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 5.7520e-04 - val_loss: 0.0072
Epoch 48/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 7.9327e-04 - val_loss: 0.0088
Epoch 49/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 6.8694e-04 - val_loss: 0.0075
Epoch 50/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.4279e-04 - val_loss: 0.0076
```

```python
# Predict
y_pred_scaled = model.predict(X_test)

# Rebuild full predicted series (merge back with rest of data for inverse transform)
dummy = np.zeros((len(y_pred_scaled), scaled_data.shape[1]))
dummy[:, 0] = y_pred_scaled[:, 0]  # only price column predicted

# Inverse transform
y_pred = scaler.inverse_transform(dummy)[:, 0]
y_actual = scaler.inverse_transform(np.concatenate([
    y_test.reshape(-1, 1),  # <-- Put price in the correct position
    np.zeros((len(y_test), scaled_data.shape[1]-1))
], axis=1))[:, 0]  # <-- Get the first column (price)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 243ms/step
```

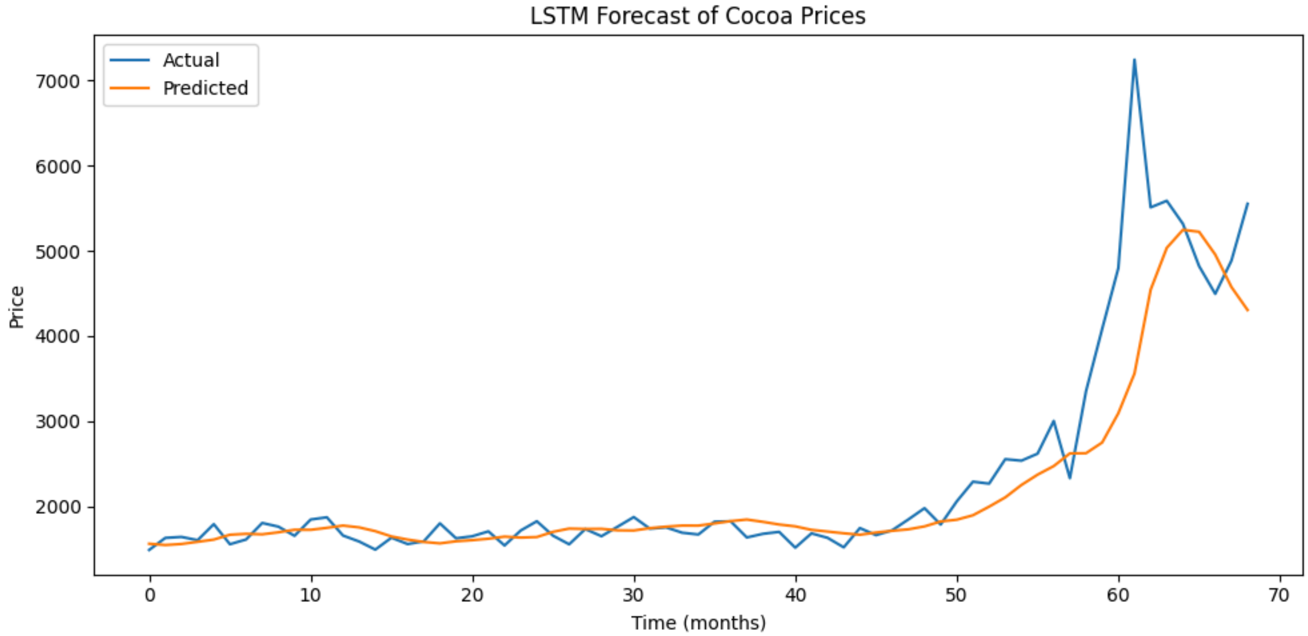
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error

rmse = np.sqrt(mean_squared_error(y_actual, y_pred))
mae = mean_absolute_error(y_actual, y_pred)
mape = np.mean(np.abs((y_actual - y_pred) / y_actual)) * 100

print(f"✅ RMSE: {rmse:.2f}")
print(f"✅ MAE: {mae:.2f}")
print(f"✅ MAPE: {mape:.2f}%")
```

```
✅ RMSE: 585.73
✅ MAE: 272.50
✅ MAPE: 8.73%
```

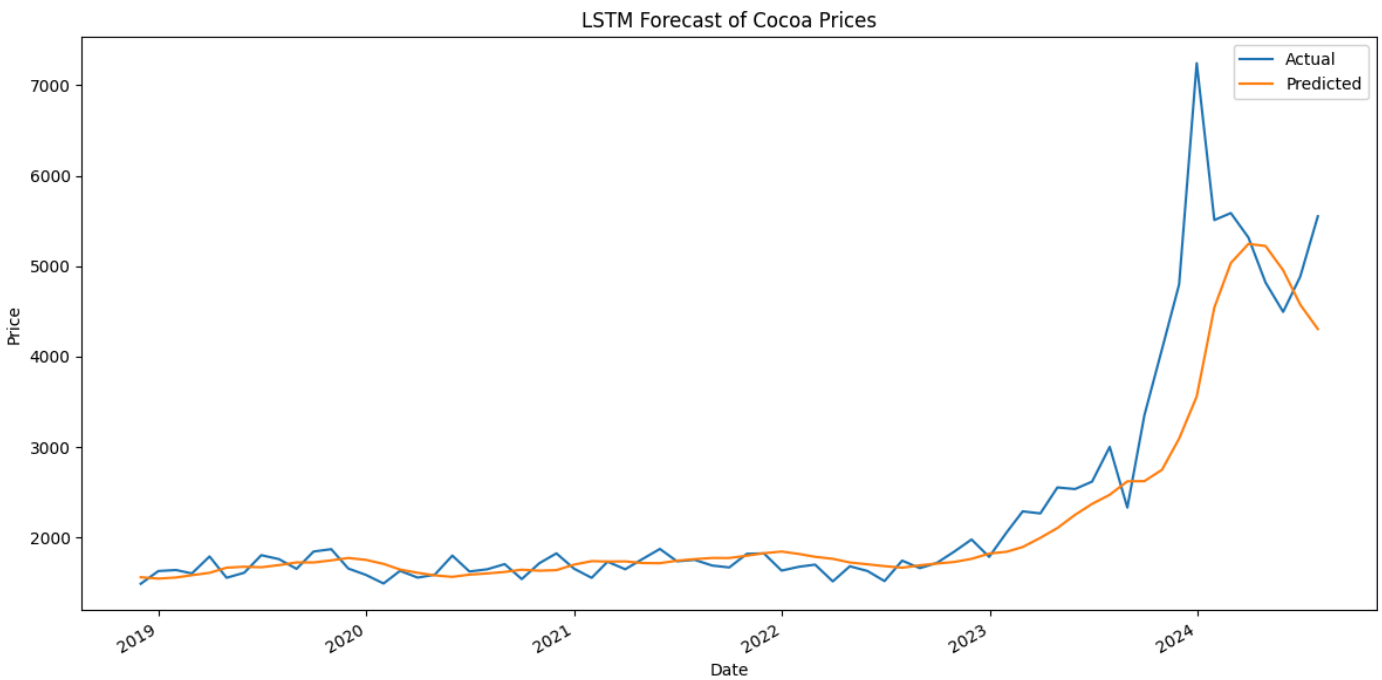
```python
plt.figure(figsize=(10, 5))
plt.plot(y_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.title("LSTM Forecast of Cocoa Prices")
plt.xlabel("Time (months)")
plt.ylabel("Price")
plt.legend()

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

```python
# Assume this is your date Series from the dataset
```

```
# Replace this with your actual date column
# For example: dates = pd.to_datetime(df['date_column'])
dates = pd.date_range(start="2018-11", periods=len(y_actual), freq='M')

# Now plot using the dates
plt.figure(figsize=(12, 6))
plt.plot(dates, y_actual, label='Actual')
plt.plot(dates, y_pred, label='Predicted')

plt.title("LSTM Forecast of Cocoa Prices")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()

# Format the x-axis to show year-month
plt.gcf().autofmt_xdate()  # Rotate date labels
plt.tight_layout()
plt.show()
```




```
<ipython-input-9-4779f9fdc595>:16: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  dates = pd.date_range(start="2018-11", periods=len(y_actual), freq='M')
```




```
print(len(y_actual))
```

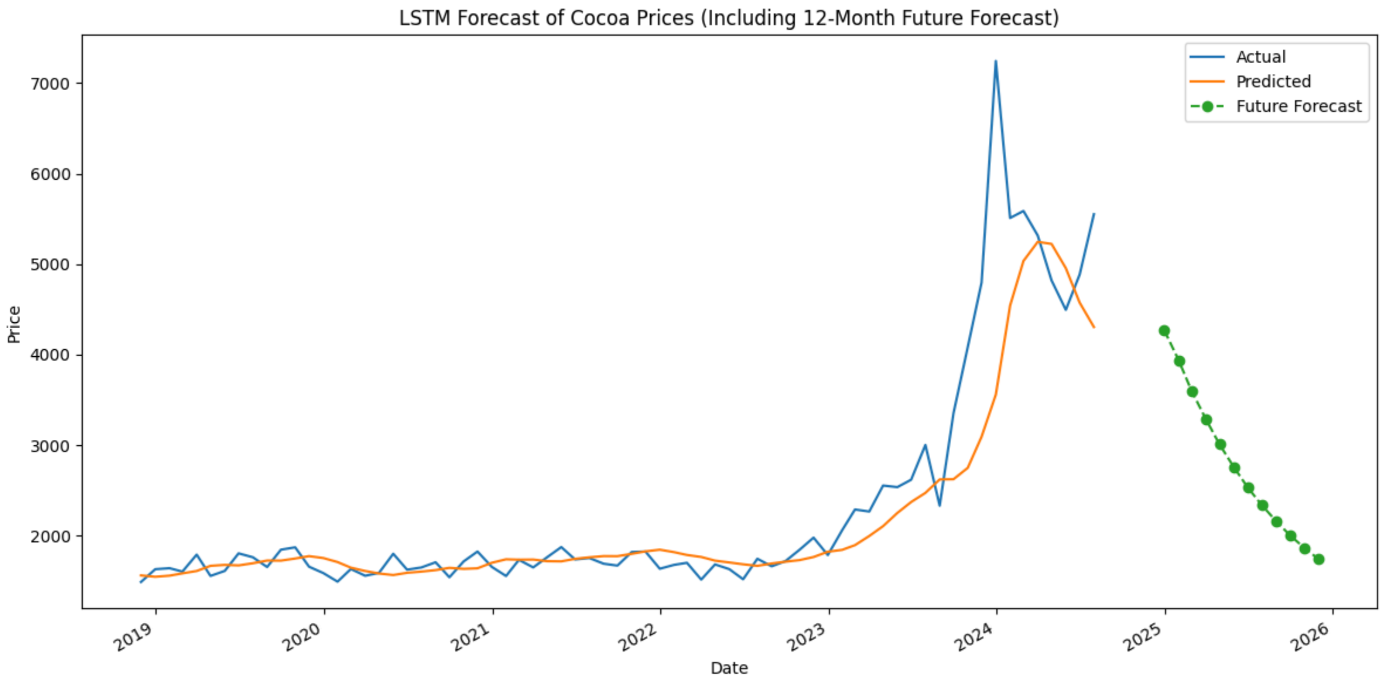
```
69
```

```
# Start from the last sequence in the dataset
last_sequence = scaled_data[-seq_len:].copy()

future_predictions_scaled = []

for _ in range(12):
    input_seq = last_sequence.reshape(1, seq_len, scaled_data.shape[1])
    pred = model.predict(input_seq, verbose=0)[0][0]

    # Build next row with predicted price, and dummy 0s for PRCP and TAVG
    next_row = np.zeros((scaled_data.shape[1],))
    next_row[0] = pred  # predicted price
```

```
        # Append prediction to list
        future_predictions_scaled.append(pred)

        # Update sequence: drop first, add new predicted row
        last_sequence = np.vstack((last_sequence[1:], next_row))

    # Reconstruct dummy for inverse transform
    dummy_future = np.zeros((len(future_predictions_scaled), scaled_data.shape[1]))
    dummy_future[:, 0] = future_predictions_scaled

    # Inverse transform to get price values
    future_prices = scaler.inverse_transform(dummy_future)[:, 0]




    # Get the last date from the original dataset
    last_y_pred_date = df.index[seq_len + split + len(y_pred) - 1]
    future_dates = pd.date_range(start=last_y_pred_date + pd.DateOffset(months=1), periods=12, freq='M')

    print(df)
    print(last_y_pred_date)
    print(future_dates)
```

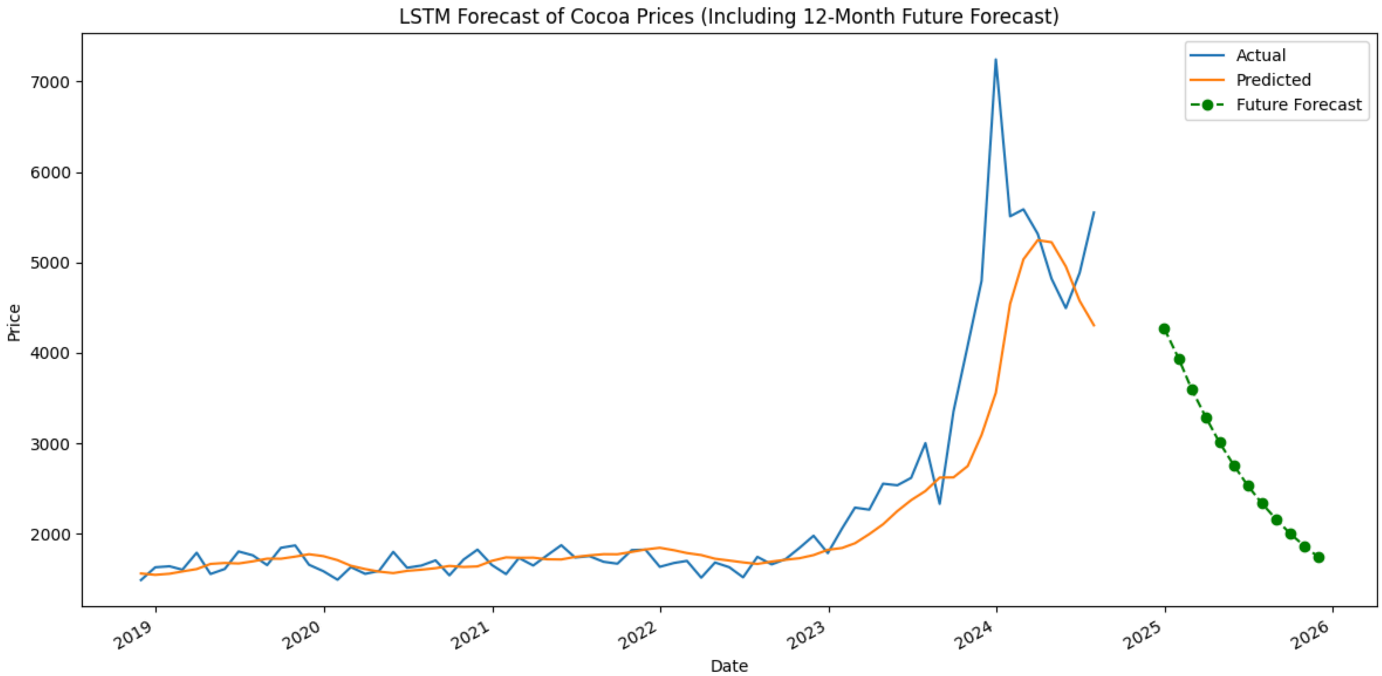
```
            Price_Monthly_Avg  Price_Monthly_Max  PRCP_Monthly_Avg  \
date
1994-10-31        1048.523448            1497.14          0.140249
1994-11-30        1053.785667            1467.82          0.098021
1994-12-31         947.974193            1446.04          0.000000
1995-01-31         995.005806            1504.70          0.000000
1995-02-28        1078.883929            1568.62          0.025294
...                       ...                ...               ...
2024-07-31        5315.684839            7687.68          0.111857
2024-08-31        4820.096774            7542.79          0.064006
2024-09-30        4495.275667            6826.34          0.096927
2024-10-31        4884.055484            7108.10          0.329627
2024-11-30        5551.087667            9099.80          0.273642

            TAVG_Monthly_Avg  TMAX_Monthly_Avg  TMIN_Monthly_Avg
date
1994-10-31         66.601574         56.919524         58.161905
1994-11-30         79.164167         82.740278         68.793585
1994-12-31          0.000000          0.000000          0.000000
1995-01-31          2.741935          0.000000          0.000000
1995-02-28         42.161012         34.556710         30.009420
...                      ...               ...               ...
2024-07-31         78.856592         84.584405         73.569759
2024-08-31         74.980914         80.495225         69.726656
2024-09-30         79.029934         86.861664         72.704153
2024-10-31         79.964631         88.084639         73.690233
2024-11-30         70.956548         77.499542         62.973471

[362 rows x 6 columns]
2024-11-30 00:00:00
DatetimeIndex(['2024-12-31', '2025-01-31', '2025-02-28', '2025-03-31',
               '2025-04-30', '2025-05-31', '2025-06-30', '2025-07-31',
               '2025-08-31', '2025-09-30', '2025-10-31', '2025-11-30'],
              dtype='datetime64[ns]', freq='ME')
<ipython-input-19-d8a85876e72e>:3: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  future_dates = pd.date_range(start=last_y_pred_date + pd.DateOffset(months=1), periods=12, freq='M')
```

```
plt.figure(figsize=(12, 6))

# Plot existing predictions with actual values
plt.plot(dates, y_actual, label='Actual')
plt.plot(dates, y_pred, label='Predicted')

# Plot future predictions
plt.plot(future_dates, future_prices, label='Future Forecast', linestyle='--', marker='o')

plt.title("LSTM Forecast of Cocoa Prices (Including 12-Month Future Forecast)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.gcf().autofmt_xdate()
plt.tight_layout()
plt.show()
print(dates)
```




```
plt.figure(figsize=(12, 6))

# Plot actual and predicted
plt.plot(dates, y_actual, label='Actual')
plt.plot(dates, y_pred, label='Predicted')

# Fix the future dates starting right after last predicted month
last_y_pred_date = df.index[seq_len + split + len(y_pred) - 1]
future_dates = pd.date_range(start=last_y_pred_date + pd.DateOffset(months=1), periods=12, freq='M')

# Plot future predictions
plt.plot(future_dates, future_prices, label='Future Forecast', linestyle='--', marker='o', color='green')

plt.title("LSTM Forecast of Cocoa Prices (Including 12-Month Future Forecast)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.gcf().autofmt_xdate()
plt.tight_layout()
plt.show()
```

```
<ipython-input-20-65b8cc85b5ca>:9: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  future_dates = pd.date_range(start=last_y_pred_date + pd.DateOffset(months=1), periods=12, freq='M')
```




```
print(dates)
print(last_y_pred_date)
```

```
DatetimeIndex(['2018-11-30', '2018-12-31', '2019-01-31', '2019-02-28',
               '2019-03-31', '2019-04-30', '2019-05-31', '2019-06-30',
               '2019-07-31', '2019-08-31', '2019-09-30', '2019-10-31',
               '2019-11-30', '2019-12-31', '2020-01-31', '2020-02-29',
               '2020-03-31', '2020-04-30', '2020-05-31', '2020-06-30',
               '2020-07-31', '2020-08-31', '2020-09-30', '2020-10-31',
               '2020-11-30', '2020-12-31', '2021-01-31', '2021-02-28',
               '2021-03-31', '2021-04-30', '2021-05-31', '2021-06-30',
               '2021-07-31', '2021-08-31', '2021-09-30', '2021-10-31',
               '2021-11-30', '2021-12-31', '2022-01-31', '2022-02-28',
               '2022-03-31', '2022-04-30', '2022-05-31', '2022-06-30',
               '2022-07-31', '2022-08-31', '2022-09-30', '2022-10-31',
               '2022-11-30', '2022-12-31', '2023-01-31', '2023-02-28',
               '2023-03-31', '2023-04-30', '2023-05-31', '2023-06-30',
               '2023-07-31', '2023-08-31', '2023-09-30', '2023-10-31',
               '2023-11-30', '2023-12-31', '2024-01-31', '2024-02-29',
               '2024-03-31', '2024-04-30', '2024-05-31', '2024-06-30',
               '2024-07-31'],
              dtype='datetime64[ns]', freq='ME')
2024-11-30 00:00:00
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.layers import Dropout

# 1. Load data
df = pd.read_csv("price_climate.csv", parse_dates=["date"])
df.set_index("date", inplace=True)

# 2. Select features
data = df[["Price_Monthly_Avg", "PRCP_Monthly_Avg", "TAVG_Monthly_Avg"]].fillna(0)

# 3. Normalize features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

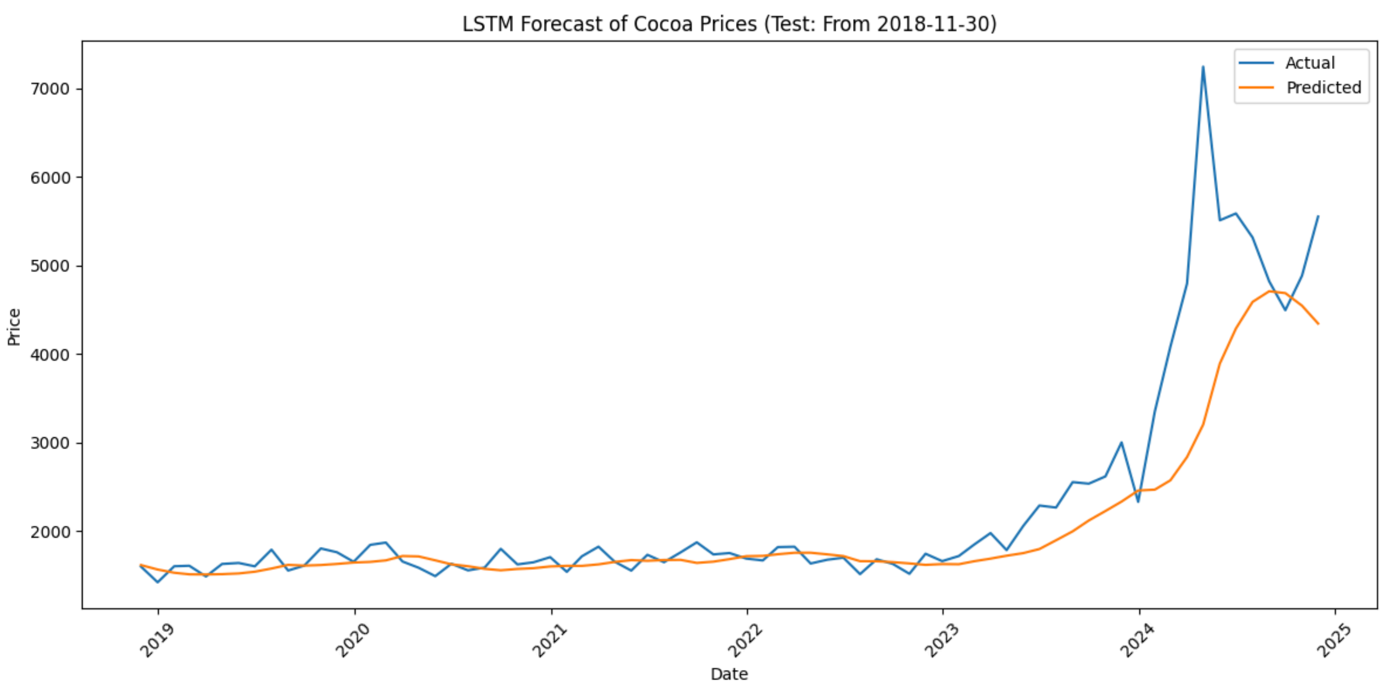
```
# 4. Create sequences WITH date tracking
def create_sequences_with_dates(data, dates, seq_len):
    X, y, y_dates = [], [], []
    for i in range(seq_len, len(data)):
        X.append(data[i - seq_len:i])
        y.append(data[i, 0])
        y_dates.append(dates[i])
    return np.array(X), np.array(y), np.array(y_dates)

seq_len = 6
dates = data.index  # datetime index
X, y, y_dates = create_sequences_with_dates(scaled_data, dates, seq_len)

# 5. Time-based split
train_cutoff = pd.Timestamp("2018-10-31")
test_start = pd.Timestamp("2018-11-30")

train_mask = y_dates <= train_cutoff
test_mask = y_dates >= test_start

X_train, y_train = X[train_mask], y[train_mask]
X_test, y_test = X[test_mask], y[test_mask]
test_dates = y_dates[test_mask]

# 6. Build LSTM model
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(seq_len, X.shape[2])),
    # Dropout(0.2),
    LSTM(32),
    # Dropout(0.2),
    Dense(1)
])
model.compile(loss='mse', optimizer='adam')
model.summary()

# 7. Train
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
                    validation_data=(X_test, y_test), verbose=1)

# 8. Predict (on test set)
y_pred_scaled = model.predict(X_test)

# 9. Inverse transform
# For prediction: construct dummy to match shape
dummy_pred = np.zeros((len(y_pred_scaled), scaled_data.shape[1]))
dummy_pred[:, 0] = y_pred_scaled[:, 0]
y_pred = scaler.inverse_transform(dummy_pred)[:, 0]

# For actual: same for y_test
dummy_actual = np.zeros((len(y_test), scaled_data.shape[1]))
dummy_actual[:, 0] = y_test
y_actual = scaler.inverse_transform(dummy_actual)[:, 0]

# 10. Metrics
rmse = np.sqrt(mean_squared_error(y_actual, y_pred))
mae = mean_absolute_error(y_actual, y_pred)
mape = np.mean(np.abs((y_actual - y_pred) / y_actual)) * 100
print(f"✅ RMSE: {rmse:.2f}")
print(f"✅ MAE: {mae:.2f}")
print(f"✅ MAPE: {mape:.2f}%")

# print(f"✅ ME (Mean Error): {me:.2f}")


# 11. Plot
plt.figure(figsize=(12, 6))
plt.plot(test_dates, y_actual, label='Actual')
plt.plot(test_dates, y_pred, label='Predicted')
plt.title("LSTM Forecast of Cocoa Prices (Test: From 2018-11-30)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_di
  super().__init__(**kwargs)
Model: "sequential_14"
```

Layer (type)	Output Shape	Param #
lstm_28 (LSTM)	(None, 6, 64)	17,408
lstm_29 (LSTM)	(None, 32)	12,416
dense_14 (Dense)	(None, 1)	33

```
 Total params: 29,857 (116.63 KB)
 Trainable params: 29,857 (116.63 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 40ms/step - loss: 0.0144 - val_loss: 0.0442
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0048 - val_loss: 0.0281
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0027 - val_loss: 0.0204
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0017 - val_loss: 0.0120
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 8.8180e-04 - val_loss: 0.0099
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.1533e-04 - val_loss: 0.0120
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 9.1629e-04 - val_loss: 0.0108
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 7.9588e-04 - val_loss: 0.0110
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.8121e-04 - val_loss: 0.0111
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 8.4610e-04 - val_loss: 0.0111
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 7.5947e-04 - val_loss: 0.0100
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 6.9998e-04 - val_loss: 0.0115
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 7.9223e-04 - val_loss: 0.0105
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 7.1261e-04 - val_loss: 0.0106
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.0835e-04 - val_loss: 0.0108
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 7.0114e-04 - val_loss: 0.0114
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 8.4284e-04 - val_loss: 0.0102
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 9.0249e-04 - val_loss: 0.0096
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 8.4466e-04 - val_loss: 0.0123
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 8.3689e-04 - val_loss: 0.0103
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 7.2705e-04 - val_loss: 0.0110
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 8.3684e-04 - val_loss: 0.0104
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 9.1567e-04 - val_loss: 0.0097
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 6.9644e-04 - val_loss: 0.0106
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.1030e-04 - val_loss: 0.0114
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 7.1658e-04 - val_loss: 0.0093
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.4638e-04 - val_loss: 0.0105
Epoch 28/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.9073e-04 - val_loss: 0.0101
Epoch 29/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.2943e-04 - val_loss: 0.0108
Epoch 30/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 7.2170e-04 - val_loss: 0.0103
Epoch 31/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.8098e-04 - val_loss: 0.0100
Epoch 32/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.1688e-04 - val_loss: 0.0096
Epoch 33/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 8.0016e-04 - val_loss: 0.0100
Epoch 34/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 6.7013e-04 - val_loss: 0.0102
Epoch 35/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.0787e-04 - val_loss: 0.0099
Epoch 36/50
```

```
Epoch 36/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 6.8388e-04 - val_loss: 0.0117
Epoch 37/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 8.2691e-04 - val_loss: 0.0093
Epoch 38/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 8.5306e-04 - val_loss: 0.0108
Epoch 39/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.0361e-04 - val_loss: 0.0097
Epoch 40/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.4772e-04 - val_loss: 0.0102
Epoch 41/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.6355e-04 - val_loss: 0.0095
Epoch 42/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.6057e-04 - val_loss: 0.0092
Epoch 43/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 7.6269e-04 - val_loss: 0.0099
Epoch 44/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 7.1755e-04 - val_loss: 0.0101
Epoch 45/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.6469e-04 - val_loss: 0.0092
Epoch 46/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.8325e-04 - val_loss: 0.0113
Epoch 47/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.8163e-04 - val_loss: 0.0093
Epoch 48/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 6.1975e-04 - val_loss: 0.0087
Epoch 49/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 5.7473e-04 - val_loss: 0.0095
Epoch 50/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 6.5123e-04 - val_loss: 0.0097
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 165ms/step
✅ RMSE: 661.60
✅ MAE: 305.83
✅ MAPE: 9.51%
```



```
me = np.mean(y_pred - y_actual)
abs_me = abs(me)
print(f"✅ ME (Mean Error): {abs_me:.2f}")
```

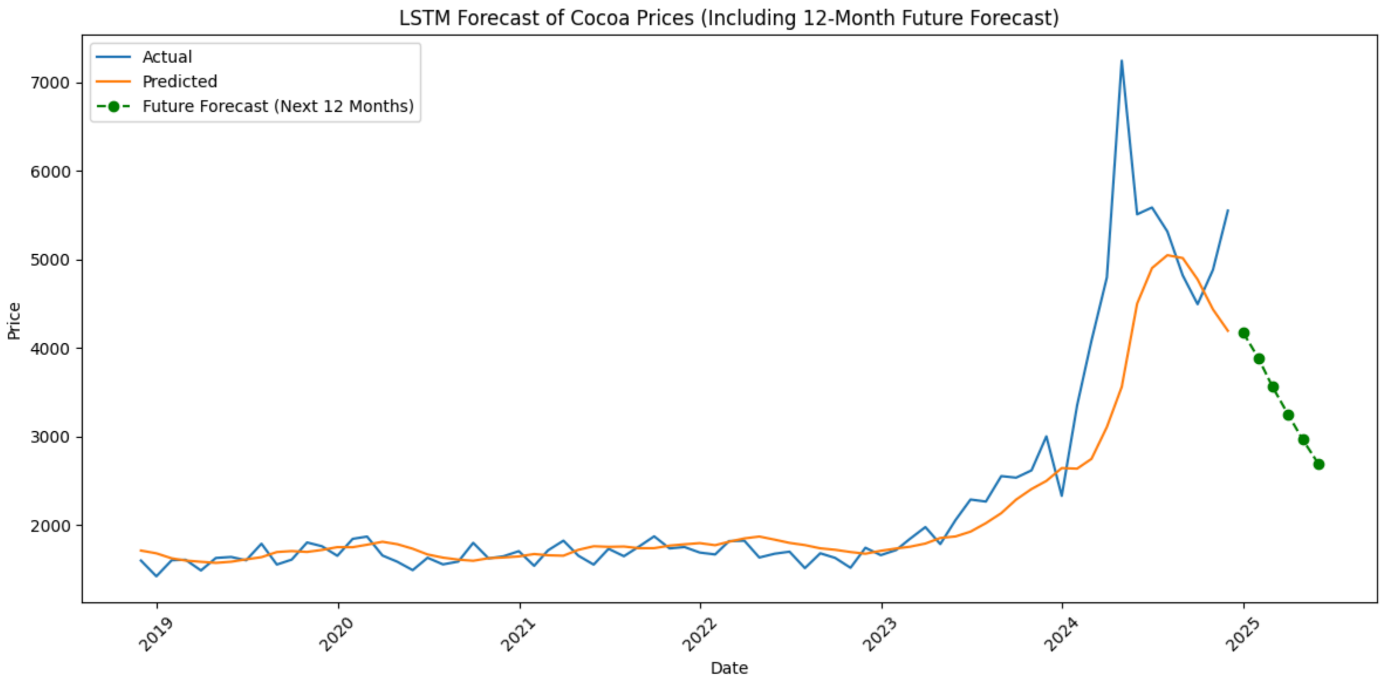
```
✅ ME (Mean Error): 260.48
```

```
# Start with the last known sequence
last_sequence = scaled_data[-seq_len:].copy()  # shape: (18, 3)

future_predictions_scaled = []

for _ in range(6):
    input_seq = last_sequence.reshape(1, seq_len, scaled_data.shape[1])
    pred = model.predict(input_seq, verbose=0)[0][0]

    # Prepare next input row: predicted price + dummy 0s for other features
    next_row = np.zeros((scaled_data.shape[1],))
    next_row[0] = pred

    # Append to sequence and slide window
    last_sequence = np.vstack((last_sequence[1:], next_row))
    future_predictions_scaled.append(pred)

# Inverse transform predicted values
dummy_future = np.zeros((6, scaled_data.shape[1]))
dummy_future[:, 0] = future_predictions_scaled
future_prices = scaler.inverse_transform(dummy_future)[:, 0]
```

```
last_known_date = df.index[-1]
future_dates = pd.date_range(start=last_known_date + pd.DateOffset(months=1), periods=6, freq='M')
```

```
<ipython-input-45-b591408ffc70>:2: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME'
  future_dates = pd.date_range(start=last_known_date + pd.DateOffset(months=1), periods=6, freq='M')
```

```
plt.figure(figsize=(12, 6))

# Plot previous predictions
plt.plot(test_dates, y_actual, label='Actual')
plt.plot(test_dates, y_pred, label='Predicted')

# Plot future predictions
plt.plot(future_dates, future_prices, label='Future Forecast (Next 12 Months)', linestyle='--', marker='o', color='green')

plt.title("LSTM Forecast of Cocoa Prices (Including 12-Month Future Forecast)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.layers import Dropout

# 1. Load data
df = pd.read_csv("price_climate.csv", parse_dates=["date"])
df.set_index("date", inplace=True)

# 2. Select features
data = df[["Price_Monthly_Avg", "PRCP_Monthly_Avg", "TAVG_Monthly_Avg"]].fillna(0)

# 3. Normalize features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# 4. Create sequences WITH date tracking
def create_sequences_with_dates(data, dates, seq_len):
    X, y, y_dates = [], [], []
    for i in range(seq_len, len(data)):
        X.append(data[i - seq_len:i])
        y.append(data[i, 0])
        y_dates.append(dates[i])
    return np.array(X), np.array(y), np.array(y_dates)

seq_len = 12
dates = data.index  # datetime index
X, y, y_dates = create_sequences_with_dates(scaled_data, dates, seq_len)

# 5. Time-based split
train_cutoff = pd.Timestamp("2018-10-31")
test_start = pd.Timestamp("2018-11-30")

train_mask = y_dates <= train_cutoff
test_mask = y_dates >= test_start

X_train, y_train = X[train_mask], y[train_mask]
X_test, y_test = X[test_mask], y[test_mask]
test_dates = y_dates[test_mask]
```